6-2014

Parameter Identification and Robust Control Applied to a Quadrotor

Ismail Sulaiman Al-Ali

United Arab Emirates University

College of Engineering

Department of Electrical Engineering

Parameter Identification and Robust Control Applied to a Quadrotor

Ismail Sulaiman Al-Ali

This thesis is submitted in partial fulfillment of the requirements for the Master of Science in Electrical Engineering degree

Under the direction of Professor Hassan Noura

June 2014

I, Ismail Sulaiman Al-Ali, the undersigned, a graduate student at the United Arab Emirates University (UAEU) and the author of the thesis titled "Parameter Identification and Robust Control Applied to a Quadrotor", hereby solemnly declare that this thesis is an original research work done and prepared by me under the guidance of Prof. Hassan Noura, in the College of Engineering at UAEU. This work has not been previously formed as the basis for the award of any academic degree, diploma or similar title at this or any other university. The materials borrowed from other sources and included in my thesis have been properly cited and acknowledged.

Student's Signature ..... Date 21.06.2014

Thesis Examination Committee:

1) Advisor: Prof. Hassan Noura

   Title: Professor, Department Chairman

   Department: Electrical Engineering

   Institution: College of Engineering, UAE University

   Signature: _____ Date: 26/6/2014

2) Co-advisor: Dr. Zaher Daboussi

   Title: Chief Technology Officer - CTO

   Department: Development and Technology – D&T

   Institution: Abu Dhabi Autonomous Systems Investments - ADASI

   Signature: _____ Date: 19.06.14

3) Member: Dr. Addy Wahyudie

   Title: Assistant Professor

   Department: Electrical Engineering

   Institution: College of Engineering, UAE University

   Signature: _____ Date: 26.6.2014

4) External Examiner: Dr. Youmin Zhang

   Title: Professor

   Department: Department of Mechanical & Industrial Engineering

   Institution: Concordia University

   Signature: _____ Date: 26-1-2014

Accepted by

Master's Program Director Dr. Abbas Fardoun

Signature: _____ Date: 26-6-2014

Dean of the College.....

Signature: _____ Date: 26/6/2014

26/6/2014

Copy____of____

Abstract

This research intended to design, analyze, and implement a robust controller for a quad rotor system and compare the designed robust controller with a proportional-integral-derivative (PID) controller. The ArduCopter platform was used as a target system with a 3DR airframe and necessary modifications, a system model, and system identification processes were executed as prerequisite steps to reach the objective.

The work in this thesis includes exploring the existing research on this topic and builds on the results presented in these previous studies to add value to the scope of quad-rotor system control. During this study, system modeling was conducted, where a near-hover non-linear model for the system was obtained and realized in Simulink. Furthermore, system identification was performed to obtain the platform parameters, which include the blade thrust coefficient, inertias, and propeller drag coefficient. The identification process was based on standalone experiments as well as flight data and the non-linear model was validated and assured to be representative of the real system. The control system was then designed with both classical PID and robust controllers. This control architecture was designed to be scalable to other platforms. The classical controller was designed analytically for the body rate loop, while root-locus plots were then used for the attitude loop. The robust controller was designed based on the H_∞ method and the augmented plant was constructed using the GS/T scheme. The existing software for the ArduCopter was modified to have a cus-

tomized logging structure and flight modes functionality, and to make it suitable to implement a robust controller in state-space form. Finally, experimental flights were conducted to tune the classical controller and test the robust controller, and to conduct robustness tests by injecting user-controlled, known disturbances in flight.

Various outcomes were reached and findings were made along the research stages. One of the outcomes reached, was to determine the effectiveness of the identification methods used, despite the shortages in the standalone experimental setup. Furthermore, the yaw torque model reused from previous studies was found to not match properly with the flight data. The drag on propeller rotation as presented in the literature is considered to be dominant over the anti-torque action. The flight data and analysis thereof in this research show that anti-torque contributes more to generating yaw torque than propeller drag.

Furthermore, a comparison between the PD controller and the robust controller was made during the experimental flights. The flight data showed that the PD controller has a good dynamical response, but lacks robustness against imbalance in actuation (or untrimmed actuators). Integrator action was added gradually over the course of a few experiments to enhance the performance without affecting the dynamical response. The tuned controller showed fairly good overall robustness when disturbances were injected manually. In comparison with the PD controller, the robust controller performed far better in terms of dynamical response and disturbances rejection, but the controller obtained is much more complex than PID controller and requires more computational time to propagate over time.

Acknowledgements

Great things can only be achieved with the support of our dear ones. Therefore, I would like to thank my supervisor, Prof. Hassan Noura, for his continuous support and wise guidance during this research. I would also like to thank Dr. Zaher Daboussi, my co-supervisor and my manager at work, for all his support and believing in me. Further, I would like to thank the United Arab Emirates University (UAEU) for providing the resources to make this research possible.

I am also very thankful to my great parents, who always encouraged and helped me to pursue engineering and continue my studies. I also owe many thanks to my loving and supportive wife, M. Al Massabi, who came into my life as a blessing and a big motivation to reach great achievements. I owe both of my parents and my wife gratitude for supporting me and pushing me forward.

Further, I want to thank my friends and former students at Stellenbosch University thank you for all your encouragement and support.

... For my dear parents and beloved wife ...

Contents

List of Figures

List of Tables

List of Symbols

Roman Symbols

α Angular acceleration (general).

$\bar{\sigma}$ Maximum singular value for a transfer function.

η The attitude vector with respect to E FoR.

γ Gyroscopic effect coefficient.

Γ_i The i^{th} generalized force in Euler-Lagrange method.

γ_{zw} Maximum singular value from input w to output z for the cost function T_{zw}.

$\hat{\tau}_{\phi,\theta,\psi}$ Estimated torque on (ϕ, θ, ψ) from derivative of angular rate.

Ω_i Angular velocity of the i^{th} propeller.

ϕ Roll angle of body.

ϕ_d Desired roll angle.

ψ Yaw angle of body.

ψ_d Desired yaw angle (heading).

ρ Air density.

τ Torque (general).

τ_ϕ Moment on x-axis.

τ_ψ Moment on z-axis.

τ_θ Moment on y-axis.

τ_D Drag torque due to rotation in air.

τ_d Desired time constant for designed body roll rate controller.

τ_{mass} Torque applied by the mass in the load cell calibration process.

τ_{mp} Time constant of motor-propeller subsystem.

θ Pitch angle of body.

θ_d Desired pitch angle.

ξ The position vector with respect to E FoR.

ξ_B The position vector with respect to B FoR.

ξ_E The position vector with respect to E FoR.

ζ Damping of a linear system.

A Cross sectional area of an object when determining the drag force.

b Thrust coefficient of the propeller.

b_{ml} Radius of oscillation in Miller's Inertia measurement experiment.

C_D Drag coefficient (dimensionless).

cmd_i Inputs commands to ArduCopter environment.(i = [1: roll, 2: pitch, 3: yaw, 4: thrust]

d Drag coefficient of the propeller angular velocity.

E_{KE} Kinetic energy.

E_{PE} Potential energy.

F_D Drag force of aircraft.

F_{cell} Force applied at the load cell.

g Gravity acceleration constant.

I Inertia (general).

I_r Inertia of propeller.

I_{ml} Object inertia in Miller's Inertia measurement experiment.

I_{xx} Body inertia on x-axis.

I_{yy} Body inertia on y-axis.

I_{zz} Body inertia on z-axis.

K_{robust} Robust controller K obtained using H_∞.

L Lagrangian term.

l Arm length between quadrotor center to motor-propeller center.

L_{cell} Length from center of rotation to loaded mass.

L_{Ixx} Length of vertical lines when measuring inertia on x-axis.

L_{Izz} Length of vertical lines when measuring inertia on z-axis.

L_{mass} Length from center of rotation to loaded mass.

L_{ml} Length of vertical lines in Miller's Inertia measurement experiment.

L_{motor} Distance of motor to center of rotation of the arm in the motor-propeller identification process.

M Actuation matrix.

m Body mass.

m_{mass} mass of the loaded-mass in the load cell calibration process.

M_{xgyro} Moment due to propeller gyroscopic effect on x-axis.

M_{xgyro} Moment due to propeller gyroscopic effect on y-axis.

p Angular rate on x-axis with respect to ξ_B.

P_i Pulse width of the signal to the i^{th} motor in unit of time $[\mu sec]$.

p_i Commanded pulse width of the i^{th} motor in percentage.

q Angular rate on y-axis with respect to ξ_B.

q_i The i^{th} generalized coordinate in Euler-Lagrange method.

r Angular rate on z-axis with respect to ξ_B.

S Sensitivity of a system.

T Complementary sensitivity of a system.

T_Δ Variation in thrust force around operating point.

T_i Thrust force produced by the i^{th} propeller.

T_t Total thrust force produced by all propellers.

T_{ml} Oscillation period in Miller's Inertia measurement experiment.

T_{op} Thrust force compensation for attitude changes.

T_{trim} Thrust force at the operating point.

U Input vector of actuators commands in terms of moments and total thrust.

u Body velocity on x-axis with respect

u_d Desired velocity on x-axis (forward) with respect to B FoR.

v Body velocity on y-axis with respect to ξ_B.

v_d Desired velocity on y-axis (sideway to right direction) with respect to B FoR.

V_i Voltage applied on the i^{th} motor.

v_{as} Aircraft airspeed.

w Body velocity on z-axis with respect to ξ_B.

w_d Desired velocity on z-axis (upward) with respect to B FoR.

W_e Sensitivity weighting function in the $S/KS/T$ scheme.

W_s Template for setting sensitivity weighting function.

W_t Template for setting complementary sensitivity weighting function.

W_u Control signal weighting function in the $S/KS/T$ scheme.

W_y Complementary sensitivity weighting function in the $S/KS/T$ and the GS/T schemes.

$W_{\bar{u}}$ Sensitivity weighting function for GS/T scheme.

W_{ml} Object weight in Miller's Inertia measurement experiment.

X X position of system on ξ_E.

x Element of x-axis on ξ_B.

X_d Desired X position on E FoR.

Y Y position of system on ξ_E.

y Element of y-axis on ξ_B.

Y_d Desired Y position on E FoR.

Z Z position of system on ξ_E.

z Element of z-axis on ξ_B.

Z_d Desired Z position (height) on E FoR.

Z_Δ Altitude variation around operating point.

Z_{op} System altitude at the operating point.

List of Abbreviations

3D	Three-dimensional.
APM	Ardu-Pilot Mega.
BEC	Battery Eliminator Circuit
CFD	Computational Fluid Dynamics.
CG	Center of Gravity.
DC	Direct Current.
DoF	Degree of Freedom.
ESC	Electronic Speed Controller.
FAI	Fdration Aronautique Internationale (World Air Sports Federation).
FoR	Frame of Reference.
GS/T	The GS/T scheme for augmented plant.
HILS	Hardware In the Loop Simulation.
LFT	Linear Fractional Transformation.
LMI	Linear Matrix Inequality.
LPF	Low Pass Filter.
LQR	Linear Quadratic Regulator.

LTI	Linear Time-Invariant system.
MCU	Micro-Controller Unit.
MEMS	Micro-Electronic Mechanical Sensor
MIMO	Multi-Input Multi-Output system.
PC	Personal Computer.
PD	Proportional/Derivative controller.
PI	Proportional/Integral controller.
PID	Proportional/Integral/Derivative controller.
PSD	Power Spectral Density.
PWM	Pulse Width Modulation.
R&D	Research and Development.
RC	Remote-Controlled.
RMS	Root-Mean-Square.
RPM	Revolutions Per Minute.
S/KS/T	The $S/KS/T$ scheme for augmented plant.
SISO	Single-Input Single-Output system.
UAV	Unmanned Air Vehicle.
US	United States of America
USB	Universal Serial Bus.
VTOL	Vertical Takeoff and Landing.
GS-PID	Gain-Scheduled PID

MPC	Model Predictive Control
MRAC	Model Reference Adaptive Control
MHE	Moving Horizon Estimation
UKF	Unscented Kalman Filter

Chapter 1

Introduction

The purpose of this chapter is to introduce the quadrotor, and give a general overview of the platform. First, this chapter gives a historical overview of previous attempts and projects to build a quadrotor system since 1921 until mid of 19th century. The flying concept of the quadrotor is also explained in this chapter, and how can this platform achieve the basic maneuvers to navigate within space. Also, the advantages and disadvantages of quadrotors is discussed in this chapter, and why this platform has become a very attractive choice for indoor applications specially. Moreover, this chapter gives an overview of recent researches relating to quadrotors. The literature covers topics relating to: modeling of the system, robust control, nonlinear control, adaptive control, fault tolerant control, and classical control.

1.1 History

The history of the development of quadrotor aircrafts began after World War I, in France, when a French engineer and helicopter designer, Etienne Oehmichen, started designing and building the first quadrotor aircraft models. He designed six different quadrotors and achieved the first successful flight with such an aircraft in 1921. The Oehmichen No.2 aircraft was an enhanced prototype of Oehmichens initial designs and consisted of a single engine, four lifting rotors, and eight variable-pitch propellers for steering and maneuvering [11]. The photo in Figure 1.1 shows Oehmichens aircraft hovering just above ground level.

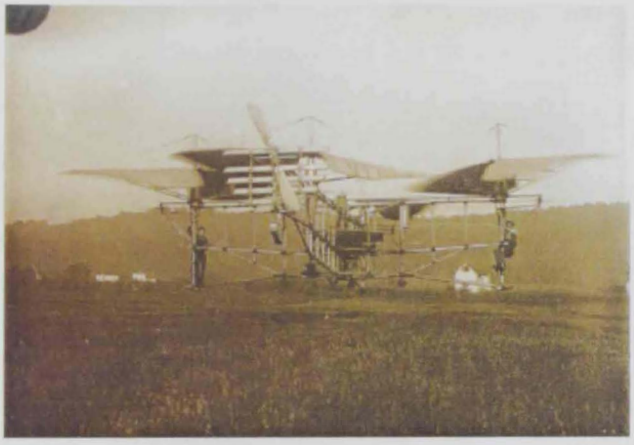


Figure 1.1: Oehmichen's quadrotor in flight (taken from [40])

In 1923, Oehmichen No.2 was capable of hovering and remaining airborne for several minutes. In 1924, Oehmichen No.2 managed to fly a distance of 360 m. This achievement broke the World Air Sports Federation (FAI) records for a helicopter flight at that time. Later that same year, Oehmichen won a prize for flying the quadrotor and carrying out a closed-circuit flight followed by a triangular trajectory, with a total distance traveled of 1 km and total flight duration of seven minutes and 40 seconds.

In parallel with Oehmichen in France, the Russian American engineer George de Bothezat was also working on a quadrotor in Ohio, in the United

States (US) of America. His quadrotor was fitted with a 170 hp Le Rhone engine [1]. This aircraft was developed on full scale and no preliminary tests were conducted with models. All the design and construction details were calculated by Bothezat based on his theory of helicopter stability.

As can be seen in Figure 1.2, the x-shaped aircraft structure designed by Bothezat consisted of four main rotors. Each rotor consisted of six variable-pitch propellers for lift generation. Further, the design consisted of two small propellers with variable pitch. The designing, building, and testing activities for this aircraft took 18 months and the US Air Service built the aircraft under De Bothezats personal supervision. The aircraft had its first successful flight in December 1922 in Ohio, and the aircraft remained hovering for one minute and 42 seconds, with a maximum height of 1.8 m above ground level.

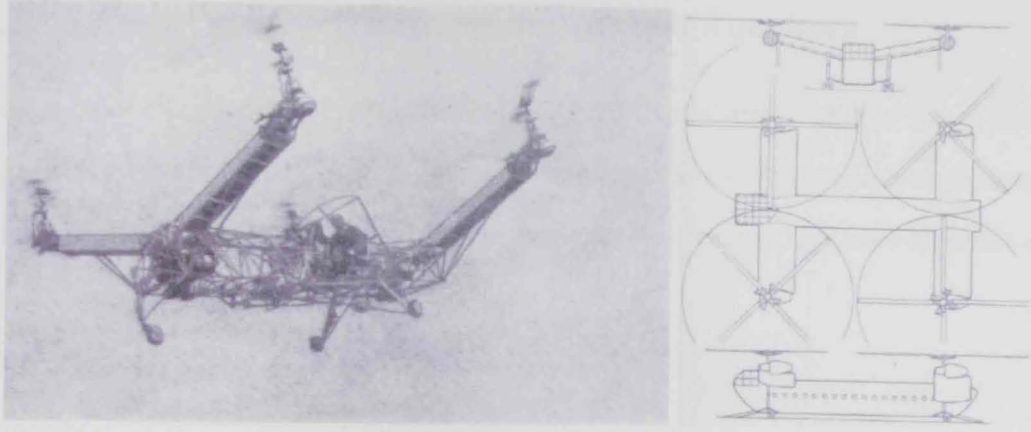
Figure 1.2: De Bothezat's quadrotor in flight (taken from [1])

After these developments, aircraft industry and research focused for few decades on tail-rotor-based rotary wing aircrafts, which developed into the conventional helicopters of today. These aircrafts have a main rotor and a tail rotor.

In 1956, a project was undertaken by Convertawings Inc. and a Model-A quadrotor aircraft was developed (shown on the left in Figure 1.3a). The team continued developing this aircraft and eventually developed a Model-E cargo quadrotor (shown on the right in Figure1.3b). This is a massive quadrotor, with a gross weight of 19050 kg, and is capable of carrying a payload of 4944

kg and reach a maximum cruise speed of 278 km/h [16].

Later, in 1958, another quadrotor project was undertaken by the Curtiss-Wright Corporation. The project was to design a quadrotor aircraft named VZ-7[1] for the US Army. However, the project was canceled as the system did not meet the armys standards, even though the system performed well in the tests.



(a) Model-A (b) Model-E

Figure 1.3: Convertawings Inc.'s quadrotor aircraft [16]

1.2 Flying Concept

The platform structure of a quadrotor consists of four sets of motor propellers as depicted in Figure 1.4. The sets are each placed an equal distance from the center of gravity (CG), and two sets of motors rotate in a clockwise direction while the other two sets rotate in a counter-clockwise direction. The sets that are opposite each other rotate in the same direction. For ease of reference, the motors are numbered from one to four for the front (1), right (2), back (3), and left (4) motors, respectively, as shown in Figure 1.4.

The actions performed by the system originate from three main sources, namely forces produced by the propellers rotational motion (which transforms into moments), anti-torque reaction from motors produced by torques, and

[1]This aircraft is now in the US Army Aviation Museum.

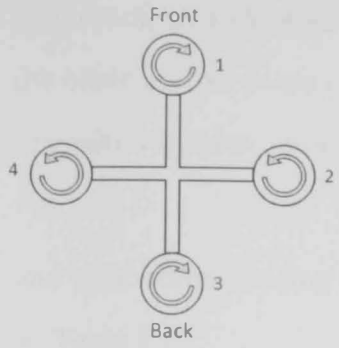


Figure 1.4: The principal structure of a quadrotor

drag from the propellers rotational motion. At equilibrium, all the forces and anti-torques are equal. Basic maneuvers can be achieved by making variations to this equilibrium and different maneuvers can be achieved by these variations. The basic maneuvers of the system are:

- Horizontal maneuvers:

  Maneuvering the quadrotor right or left can be achieved by commanding a differential angular speed between the right and left (2 and 4) motors, while leaving motors 1 and 3 unchanged. Increasing the angular speed of motor 2 while decreasing it for motor 4 causes the system to maneuver right. In order to maneuver left, the opposite should be applied (decrease the speed of motor 2 and increase the speed of motor 4). The same procedure also applies for forward and backward maneuvers by implementing the same variations on the angular speed of motors 1 and 3.

- Vertical maneuvers:

  Vertical motion can be achieved by implementing variation in all the motors angular speeds. Increasing the angular speed of all the motors equally causes the aircraft to climb, while the opposite causes the aircraft to descend.

- Turning:

  Turning right and left can be achieved by implementing variation to the total amount of anti-torque and propeller drag. Increasing the angular

speed of the motors that rotate in a clockwise direction and decreasing the angular speed of the other motors cause the system to turn counter-clockwise (left). The opposite variation causes the system to turn clockwise (right).

These basic maneuvers and their corresponding variations of motors angular speeds are summarized in Table 1.1:

Maneuver	Front Motor (1)	Right Motor (2)	Back Motor (3)	Left Motor (4)
equilibrium	x	x	x	x
Forward	↓	x	↑	x
Backward	↑	x	↓	x
Right	x	↓	x	↑
Left	x	↑	x	↓
Climb	↑	↑	↑	↑
Descend	↓	↓	↓	↓
Turn Right	↓	↑	↓	↑
Turn Left	↑	↓	↑	↓

Table 1.1: Quadrotor basic maneuvers

Classification of this platform could be done in different ways depending of the selected criteria. According to [4], aircrafts in general could be classified based on their flying principals. According to this classification, quadrotors would be powered vertical take-off and landing (VTOL) vehicles that are heavier than air. There is other literature that classifies aircrafts based on endurance and maximum altitude. According to such a classification, quadrotors are described as micro vehicles with an endurance of less than 30 minutes and maximum altitude of about 100 m. In terms of application quadrotors could be used for military, civilian, search and rescue, and research and development purposes.

1.3 Platform Evaluation

There are different factors that make the quadrotor platform favorable over other platforms for this research. One of the main factors is its simple mechan-

ical design and simple mechanical linkages that result from the fact that the blades have fixed pitch and thrust is varied by altering the rotational speed of each blade. Another advantage this mechanical simplicity adds is that such a platform is easily maintained. Another major factor that makes this platform favorable is the possibility of enclosing the four small blades within the airframe which will keep them protected when the platform collides with an obstacle in the environment. This is a major factor that makes quadrotors favorable for urban and indoor applications. Another advantage is that motors opposing each other rotate in the same direction, which, in turn, cancels out the gyroscopic effects and aerodynamic effects. As a result, modeling and analysis of the platform becomes easier. On the other hand, quadrotors have the risk of losing equilibrium where in the case of a single motor failure this will result in an unavoidable crash. However, the advantages outweigh this single disadvantage and quadrotors are very attractive for indoor applications and research and development. Therefore, they are suitable for this study.

1.4 Literature Review

Over more than a decade ago, many research groups and universities engaged in researching and developing quadrotor systems on a micro scale. The feasibility of developing this type of aircraft on a micro scale is due to the improvement in technology over the past two decades, mainly batteries capacity-to-weight ratio, the performance of electric motors, and the improvement of micro-electromechanical systems (MEMS) technology. These factors allowed quadrotor systems such as OS4 and STARMAC to be developed in [7] and [19] . These two platforms (along with other platforms) were the starting point for research and development in different fields, like fault tolerant control, robust control, adaptive control, neuro-based control, fuzzy logic-based control, and modeling/identification and validation of developed models.

One of the earliest and most significant efforts in quadrotor research and

development is the work of Samir Bouabdallah on the OS4 quadrotor system. In his article, [5], the flying principle of quadrotors is classified and evaluated among other flying principals and aircrafts. Moreover, the modeling of the OS4 platform is discussed and studied, including its airframe and rotor dynamics, and the classical control approach is tested and studied in simulation for the OS4. The research that he reports on in [6] focused more on controlling quadrotors and the results of a full autonomous flight that was achieved are presented. A linearized model of the full non-linear model of the system was derived by making some fair assumptions. Further, a comparison of two known controllers, namely the proportional-integral-derivative (PID) and linear-quadratic regulator (LQR) controllers, was made. In the LQR controller, the Riccati equation was solved to calculate the controller gain matrix. This was done using two methods, namely the Pearson method and the Sage-Eisenberg method. For testing and validation, the OS4 test bench (which is a 3DoF locked bench) was used to test and conduct experiments on the PID controller and the two LQR controllers obtained from the two methods.

The research presented in [4] and [7] later offered more accurate system model. These system models included more dynamics in forward and sideway flights. Then, a backstepping control technique was applied to stabilize the system and track a desired commanded attitude. The backstepping controller was combined with the PID technique to result in an integral backstepping controller. The integral backstepping controller performed better than the standard backstepping controller in terms or disturbance rejection.

Another series of research in controlling quadrotor systems was carried out in [26] in which a feedback linearization control was used with compensation for wind disturbances. The tracking and stability of the closed-loop system using this controller was proved using the Lyapunov stability theory. Later, in [27], the feedback linearization control was combined with linear GH_∞ in order to achieve mixed robust control. The robust feedback linearization was

based on the Sobolev norm. This new approach considers actuator saturation and constrained state-space output. The GH_∞ is applied to avoid explicitly specifying robustness and to maximize take the maximum guarantee for robustly and properly controlling the real system. This control technique was tested for disturbance rejection by injecting aerodynamic forces and moments. The closed-loop system maintained satisfactory tracking with rejection of aerodynamic forces and disturbances in presence of 20% uncertainty. In case of injecting aerodynamic moments as disturbances and with similar uncertainty, a better tracking was noticed for height and heading than the X and Y positions.

The research on feedback linearization control was continued by the same author, presented in [28], and a sliding mode observer for state estimation and perturbation estimation was studied. A disturbance parameter estimator was added in order to enhance the closed-loop system. This estimator was needed due to the non-linear disturbances in the transformation between the inner and outer-control loops. Another reason for adding this estimator was the fact that there existed non-vanishing terms in the closed-loop system.

The feedback linearization-based controller was also studied in [3] and [25]. In [3], a high-order sliding mode observer was used for state estimation and external disturbances estimation. This type of observer was used due to its insensitivity to unknown inputs and to finite-time convergence. In [25], the research focused on using a dynamic feedback controller to create a linear closed-loop system and also focused on the systems stability and robustness to external disturbances (e.g., wind or turbulences) as well as parametric uncertainties. The limitation most of the research was facing when using feedback linearization techniques was the requirement for measuring all the states. In order to avoid the impact of linearization and losing the non-linear dynamics in the process, [25] proposed using non-linear control based on Lyapunov functions to avoid the linearization process.

The research on feedback linearization and integral backstepping control

techniques continued in [41] and [15]. An inner-loop attitude controller was designed based on feedback linearization in [41], with a classical PID for trajectory tracking of pre-planned paths. This research was conducted on the R&D unmanned aerial vehicle (UAV) system, Qball-X4. The scope of the research was fault tolerant control, in which the controllers of the system were reconfigured when a failure was detected. A constrained optimization algorithm was evaluated on Qball-X4 for a fixed-point and a partial-loss failure case.

The research presented in [15] was on using integral backstepping for robust attitude and position control for an indoor quadrotor system designed for monitoring and exploration. This study defined and divided the system into three functional layers, namely navigation, flight control, and motor control. Further, a full dynamic model was presented in [14] for a commercial quadrotor, in which a sliding mode controller was tested within a simulation environment. In this research, relatively fast actuation was assumed and external disturbances from wind gusts were ignored. The research was limited to simulation and motivated the validation of the controller through experimental flights.

Another series of studies in the field is presented in [37], where a new quaternion-based feedback controller was used to stabilize the systems attitude. This work presented two controllers, a PD2 feedback structure controller and a model-independent PD controller. The former is based on compensation for Coariolis and gyroscopic torques. It consists of a proportional term (vector quaternion), and two derivative terms (airframe angular velocity and vector quaternion velocity). The second controller does not consider Coriolis and gyroscopic torques. The model-independent proportional derivative (PD) controller provided global asymptotic stability. This research was followed by [8], where a robust non-linear proportional-integral (PI) controller was used for attitude stabilization. The control approach was to combine backstepping and non-linear robust PI control and to have a switching function on the non-linear

integration. This control approach achieved robustness through non-linear design. More work inresearch by the same author on control in general and not specifically on to quadrotors for the same author can be found in [39] [38].

The STARMAC quadrotor system was another platform that resulted from a series of research projects in the field. In [19] and [20] the aerodynamics of the STARMAC system were studied and modeled for higher speeds than hover flights. Three main aerodynamic effects were investigated, namely horizontal velocity, angle of attack, and airframe design. These three effects were investigated in terms of their impact on attitude and attitude stabilization and control. The validation and analysis were based on static measurements and flight data collected from STARMAC II. The classical control techniques were found to be inadequate for higher speed control. The research on this system continued in [21] in the development of STARMAC and a testbed for quadrotors was used to test novel algorithms for autonomous operations of a fleet of quadrotors. The research was focused on algorithms and techniques for trajectory generation and tracking. The purpose of the research was to use space-indexed waypoints with guaranteed planned trajectory feasibility. The accuracy achieved for path tracking was 10 cm for indoor flights, and 50 cm for outdoor flights.

In terms of model-less approaches in quadrotor control, a neuro-based controller hierarchy is presented in [33] for a micro quadrotor. The neuro-based controller is intended to be adaptive and able to overcome the shortcomings of model-based controllers. A two-stage control architecture is also used where a position controller (outer-loop) is providing desired attitude to the attitude controller (inner-loop). The neuro-controller was found to be much faster than the basic PID controller in terms of recovering from disturbances. Moreover, the neuro-controller showed better performance in the presence of sensor and actuator noise.

Another attempt to study neuro-based controllers was done in [29] where

an adaptive neural network control design was studied. The research was concerned with modeling inaccuracy and disturbances affecting the closed-loop system. A few adaptive approaches were evaluated and compared, such as dead zone and e-modification approaches. The newly studied method performed better in tracking, weight drift, as well as large oscillations.

In terms of fuzzy-based control, the research in [12] focused on stabilization and control of a quadrotor system using a robust, adaptive fuzzy controller. The research was intended to overcome the problem that occurs when using traditional methods, namely drift in the center of the membership functions. This phenomenon is common when persistent oscillations are present in the input. This research proposed and tested an alternative adaptation process to prevent drift, by guiding the adaptation process through a set of alternating member functions. The membership functions used were limited to Gaussian membership functions. This proposed method was compared to the e-modification method, which was found to be sacrificing performance in order to avoid drift. The proposed method was found to avoid drift successfully without affecting performance as it drives the centers of the membership functions to a valid alternative that is not necessarily zero (unlike e-modification).

In terms of fault tolerance control field, the work presented by Prof. Zhang in [22] uses Model Predictive Controller (MPC) for fault tolerant control purpose. The work investigated in nonlinear parameter estimation by comparing two estimation algorithms to perform online actuator fault estimation. These two algorithms are, Moving Horizon Estimator (MHE) and Unscented Kalman Filter (UKF). The MPC worked well in presence of fault in the system. Furthermore, the convergence of the estimated parameter was found to be faster using MHE when compared to UKF, on the other side, the MHE algorithm was found to require more computational power. Another research work is presented by the same author in [23] where a Qball-X4 platform was used to compare two controllers in trajectory tracking in presence of fault. The two

controllers are: Gain-Scheduled PID (GS-PID) and Model Reference Adaptive Control (MRAC). The presented work in [23] showed that a PID which is designed for fault-free case is not suitable to control the system in presence of fault. The extended PID controller through gain scheduling as well as MRAC controller performed well in trajectory tracking in case of fault presence. The research also makes proposal for fault detection and diagnosis scheme to be combined with GS-PID. The proposed system is expected to achieve an entire active fault tolerant GS-PID.

Other research projects have also been conducted in the field of quadrotor system modeling and control. In [31] a robust distributed controller for a formation flight of a quadrotor fleet was studied. The research added the requirement for formation performance to the design of the formation controller. The requirements were specified using mixed sensitivity and the stability of the formation was ensured by the synthesis method in the presence of arbitrary switches in the communication topology.

In [30] a robust internal-loop compensator (RIC) was investigated. The RIC was based on the disturbance compensation and vision localization techniques. The disturbances were regarded to be : inaccurately model the system and sensor noise. In [17] a classical PID was used for the control loops of the quadrotors control system. The research mainly focused on the inner-loop control and the closed-loop system was tested in simulation with 3D visualization on FlightGear. In [34] robust control against disturbances was investigated. Robustness in the control system was achieved by introducing a disturbance observer to compensate for disturbances. The observer was found to successfully estimate the disturbances and compensate for them.

Chapter 2

System Model

The purpose of this chapter is to present the model obtained for the selected quadrotor platform. A high level description of the quadrotor model is discussed in this chapter, as well as the decomposition of the model into subsystems. The axis system is also defined in this chapter, a the transformation between the frame of references is derived and presented. Furthermore, the model of the system motion using Euler-Lagrange approach is given in this chapter. Finally, the Simulink realization of the complete nonlinear model is presented in this chapter in a hierarchical manner.

2.1 Model Overview

The overall system model is composed of three main parts as shown in Figure 2.1, which are: Arducopter specific input transformation, motor and propeller dynamics, and airframe dynamics. First, the system takes the input commands for the actuators and converts them into Pulse Width Modulation (PWM) signals that feed into the motor/propeller system. The motor/propeller system then generates forces and torques according to the given PWM commands. When these forces and torques are applied to the systems airframe, the airframe will exhibit a certain attitude and position behavior. Figure 2.1 shows the composition of the ArduCopter system.

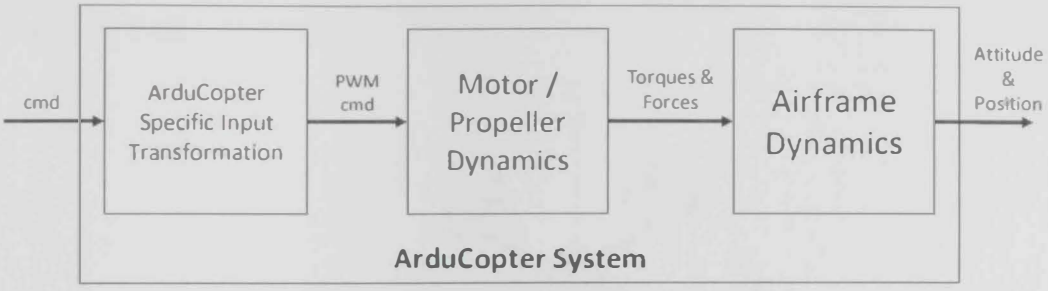


Figure 2.1: Model overview of the ArduCopter system.

The first block in the figure, the ArduCopter-specific input transformation, is added as a specific block for the ArduCopter system. The team that built and designed the system designed it with a certain level of abstraction of the commands to the airframe. Hence, the input commands for ArduCopter environment (*cmd*) needs to be transformed into the standard PWM signal. This transformation is linear and will be discussed in the Implementation and Experimentation chapter (Chapter 6).

The second block, the motor/propeller dynamics, is a common part of any quadrotor system. It takes as an input the commands for the motors and produces as an output the torques and forces acting on the body frame of reference (FoR). This block encapsulates the dynamics of the motors and propellers. The output torques and forces are then fed to the third block,

the airframe dynamics block, which describes the airframe dynamics in air. The airframe dynamics block will take forces and torques as an input and will produce the aircrafts position and velocity vectors $(\xi, \dot{\xi})$, as well as attitude and angular rates vectors $(\eta, \dot{\eta})$. In this model, the airframe dynamics and the generic six-degrees-of-freedom (6DoF) Euler model are included in the same block.

2.2 Axis System

For a rigid body moving in 3D space, there are usually two FoRs, as can be seen in Figure 2.2. One global FoR is the earth, E, FoR and the other is the body, B, FoR.

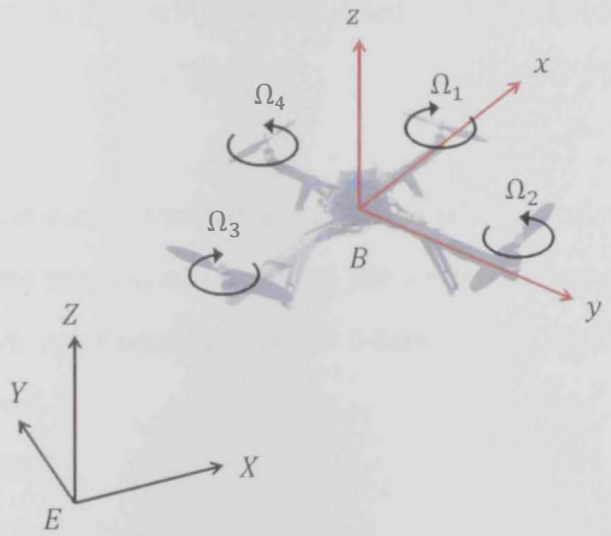


Figure 2.2: Coordinate System and axis definition of quadrotor in 3D space.

When considering the following vectors:

$\xi_E = \begin{bmatrix} X & Y & Z \end{bmatrix}^T$ is the position vector with respect to E

$\xi_B = \begin{bmatrix} x & y & z \end{bmatrix}^T$ is the position vector with respect to B

$\eta = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T$ is the attitude vector on E, also called the Euler angle vecto

A common transformation used to transform from ξ_E to ξ_B and vice versa

is the rotation matrix, R, which is defined as:

$$\xi_E = R(\phi, \theta, \psi) \times \xi_B \tag{2.1}$$

Where:

$$\begin{aligned} R &= R_z(\psi) \times R_y(\theta) \times R_x(\phi) \\ &= \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \times \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \times \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \\ &= \begin{bmatrix} C(\psi)C(\theta) & C(\psi)S(\theta)S(\phi) - S(\psi)C(\phi) & C(\psi)S(\theta)C(\phi) + S(\psi)S(\phi) \\ S(\psi)C(\theta) & S(\psi)S(\theta)S(\phi) + C(\psi)C(\phi) & S(\psi)S(\theta)C(\phi) - S(\phi)C(\psi) \\ -S(\theta) & C(\theta)S(\phi) & C(\theta)C(\phi) \end{bmatrix} \end{aligned}$$

Where:

$R_x(\phi)$ is the rotation matrix about the x-axis with an angle of ϕ

$R_y(\theta)$ is the rotation matrix about the y-axis with an angle of θ

$R_z(\psi)$ is the rotation matrix about z-axis with an angle of ψ

$C(.)$ is cos(.)

$S(.)$ is sin(.)

Hence, the velocities $\begin{bmatrix} u & v & w \end{bmatrix}$ on body FoR, B, is transformed into earth FoR, E, using (2.4):

$$\dot{X} = uC(\psi)C(\theta) + v(C(\psi)S(\theta)S(\phi) - S(\psi)C(\phi)) + w(C(\psi)S(\theta)C(\phi) + S(\psi)S(\phi)) \tag{2.2}$$

$$\dot{Y} = uS(\psi)C(\theta) + v(S(\psi)S(\theta)S(\phi) + C(\psi)C(\phi)) + w(S(\psi)S(\theta)C(\phi) - S(\phi)C(\psi)) \tag{2.3}$$

$$\dot{Z} = u(-S(\theta)) + v(C(\theta)S(\phi)) + w(C(\theta)C(\phi)) \tag{2.4}$$

This expression describes the velocity of the airframe in the earth-fixed frame, ξ_E. In the literature all research groups ([4] and [9]) followed the same approach for the kinematics model.

2.3 Modeling using the Euler-Lagrange Approach

Two approaches exist to obtain the airframe attitude dynamics. One way to obtain it is by applying Newtons law in which the sum of all the forces and moments are considered. This requires extensive mathematical manipulations, especially if a dynamical system is considered in 3D space. Another approach that can followed, as was done by [5] and [4], is using the Euler-Lagrange equation. In this research only the Euler-Lagrange method is considered. The Lagrangian equation used to derive the equation for a systems motion is:

$$L = E_{KE} - E_{PE} \tag{2.5}$$

Where E_{KE} is the kinetic energy of the system and E_{PE} is the potential energy of the system. The expression for each of these types of energy, E_{KE}

and E_{PE} , were obtained from [4]:

$$E_{KE} = \frac{1}{2} I_{xx} (\dot{\phi} - \dot{\psi} S(\theta)^2 + \frac{1}{2} I_{yy} (\dot{\theta} C(\phi) + \dot{\psi} S(\phi) C(\theta)^2 + \frac{1}{2} I_{zz} (\dot{\theta} S(\phi) - \dot{\psi} C(\phi)^2 \tag{2.6}$$

$$E_{PE} = \int x dm(x)(-gs\theta) + \int y dm(y)(gS(\phi)C(\theta)) + \int z dm(z)(gC(\phi)C(\theta)) \tag{2.7}$$

The Euler-Lagrangian equation is defined as:

$$\Gamma_i = \frac{d}{dt} \left(\frac{\partial L}{\partial \dot{q}_i} \right) - \frac{\partial L}{\partial q_i} \tag{2.8}$$

Solving the system of equations, (2.8), results in the equations of motion, (2.9), where $\dot{q}_i$ is the generalized coordinates and Γ_i is the generalized forces. The solution to the given system of equations (see [4] for a detailed derivation) is:

$$\begin{aligned} I_{xx} \ddot{\phi} &= \dot{\theta} \dot{\psi} (I_{yy} - I_{zz}) \\ I_{yy} \ddot{\theta} &= \dot{\phi} \dot{\psi} (I_{zz} - I_{xx}) \\ I_{zz} \ddot{\psi} &= \dot{\phi} \dot{\theta} (I_{xx} - I_{yy}) \end{aligned} \tag{2.9}$$

The outcome of solving the differential equation, (2.8), captures the conservative moments of the system, which are the gyroscopic effects of rotating the system airframe in 3D space. The mathematical model that describes the systems behavior consists of a conservative part described in these equations and includes a non-conservative part which results from the actuators action. The action of the propellers on the airframe is a thrust vector lifting the airframe, three moments causing angular rates, and a gyroscopic effect which results

from the propellers rotation. The thrust force is defined as:

$$T_t = b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \tag{2.10}$$

The three non-conservative moments produced by the propellers rotation action are the pitch moment, the roll moment, and the yaw moment)

$$\begin{aligned} \tau_\phi &= bl(\Omega_4^2 - \Omega_2^2) \\ \tau_\theta &= bl(\Omega_1^2 - \Omega_3^2) \\ \tau_\psi &= d(\Omega_1^2 + \Omega_3^2 - \Omega_2^2 - \Omega_4^2) \end{aligned} \tag{2.11}$$

The gyroscopic effect which results from the propellers rotation is:

$$\begin{aligned} M_{xgyro} &= I_r q\gamma \\ M_{ygyro} &= I_r p\gamma \end{aligned} \tag{2.12}$$

The mathematical model that describes the quadrotor dynamics could be summarized as in (2.13). This model is obtained by considering:

1. The body gyroscopic effect that results from body rotation (2.9);
2. The non-conservative moments that result from the actuators action. (2.11); and
3. The gyroscopic effect that results from the propellers rotation. (2.12)

$$\begin{aligned} \dot{p} &= \frac{I_{yy} - I_{zz}}{I_{xx}} qr + \frac{I_r}{I_{xx}} q\gamma + \frac{\tau_\theta}{I_{xx}} \\ \dot{q} &= \frac{I_{zz} - I_{xx}}{I_{yy}} pr + \frac{I_r}{I_{yy}} p\gamma + \frac{\tau_\phi}{I_{yy}} \\ \dot{r} &= \frac{I_{xx} - I_{yy}}{I_{zz}} pq + \frac{\tau_\psi}{I_{zz}} \end{aligned} \tag{2.13}$$

The mathematical model (2.13) describes the attitude dynamic response. The mathematical model that describes the translational response of the sys-

tem is obtained by considering the forces acting on the system. The only forces that were considered for this study are thrust, T_t, and weight, mg. Others, such as [4], have considered hub forces and body drag forces to achieve a higher fidelity model, but for this research a simple model was sufficient. If T_t and w are considered and the transformation of T_t from ξ_B to ξ_E is applied, the translational acceleration can be obtained as in (2.14):

$$\begin{aligned}
\ddot{X} &= \frac{T_t}{m}(\sin(\psi)\sin(\phi) + \cos(\psi)\sin(\theta)\cos(\phi)) \\
\ddot{Y} &= \frac{T_t}{m}(-\cos(\psi)\sin(\phi) + \sin(\psi)\sin(\theta)\cos(\phi)) \\
\ddot{Z} &= \frac{T_t}{m}\cos(\theta)\cos(\phi) - g
\end{aligned} \tag{2.14}$$

2.4 Motor-Propeller Model

The motor-propeller subsystem is composed of three main components: A brushless DC motor, a brushless electronic speed controller (ESC), and a propeller mounted on the motor shaft. These three components can be modeled and analyzed separately. However, for this research, all these components were combined into a single model. The reasons for combining all these components into a single model are:

1. The motor model is unnecessary complex for the purpose of this research, non-linear, and includes dynamics much faster than the system dynamics; and

2. The motor (with the propeller mounted) is controlled by a closed-loop speed controller, the ESC, meaning the ESC will maintain the motor angular velocity based on an input reference.

There are two parts in the motor-propeller model, namely the dynamics model and the static transformation.

The dynamics model is a first-order system with a time constant as shown below. The time constant captures the response time the ESC takes to drive

the motor angular velocity from one state to another. This time constant changes according to the ESC design as well as the combined inertia of the motor rotating shell and the inertia of the propeller.

$$H_{mp}(s) = \frac{1}{\tau_{mp}s + 1} \tag{2.15}$$

The static transformation is simply a mapping function between ESC input into reference angular velocity for the motor. The force generated by the rotating propeller is modeled as:

$$T_i = b\Omega_i^2 \tag{2.16}$$

2.5 Model Summery

If the quadrotor system is considered as a single block that describes the real platform, then this block will have four inputs and at least six outputs (this could go up to 12 outputs if the derivatives are considered), see Figure 2.3. The inputs are cmd_1,cmd_2,cmd_3,and cmd_4 and they are the input signals to the motor software layer in the ArduCopter environment. These input commands are: pitch command, roll command, yaw command, and total thrust. The output, ξ, is the position vector and is equal to $\xi = \begin{bmatrix} x & y & z \end{bmatrix}^T$, while the attitude vector (orientation), η, is equal to $\eta = \begin{bmatrix} \phi & \theta & \psi \end{bmatrix}^T$. Derivatives of both of these vectors could sometimes be included as a system output to end up with four output vectors: a position vector, ξ, an attitude vector, η, a translational speed vector, $\dot{\xi}$, and an attitude rate vector, $\dot{\eta}$.

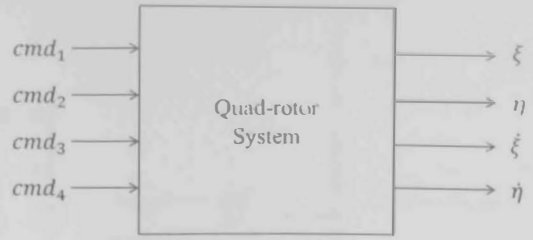


Figure 2.3: Quadrotor system

2.6 Model Realization in Simulink

2.6.1 Input Transformation

The input to the complete system was initially transformed from ArduCopters specific environment into an actual physical PWM signal - as shown in Figure fig:ModelOverview -. This transformation was identified by inspecting the source code developed and used by the original developers of the platform. The transformation from the ArduCopters environment into the PWM signal to the brushless speed controller was calculated by:

$$\begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = \frac{1}{100} \begin{bmatrix} 0 & 1 & 1 & 1 \\ -1 & 0 & -1 & 1 \\ 0 & -1 & 1 & 1 \\ 1 & 0 & -1 & 1 \end{bmatrix} \begin{bmatrix} cmd_1 \\ cmd_2 \\ cmd_3 \\ cmd_4 \end{bmatrix} - \begin{bmatrix} 0.5 \\ 0.5 \\ 0.5 \\ 0.5 \end{bmatrix} \tag{2.17}$$

Where p_n is pulse width expressed as a percentage. However, it was scaled down to $\pm 50\%$ instead of $[0, +100\%]$. The actual PWM signal in terms of pulse width could be obtained using the following equation:

$$P_j = p_j(P_{max} - P_{min}) + \frac{P_{max} + P_{min}}{2}, j = 1, 2, 3, 4 \tag{2.18}$$

The realization of the transformation is shown in Figure 2.4. The gain block and the constant block contain the matrices presented earlier in the equation and the function block contains the conversion of the PWM signal from the percentage range into the actual pulse width.

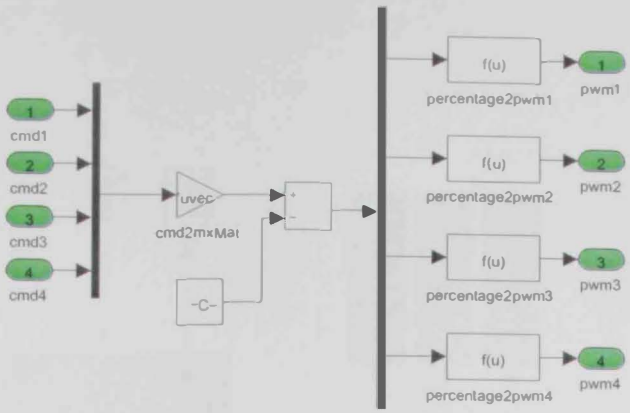


Figure 2.4: Input transformation (from ArduCopter-specific into standard PWM)

The dynamics model for the quadrotor system discussed in the previous section was modeled and tested in the MATLAB/Simulink environment. The blocks in Figure 2.5 represent the top-level block diagram in the system and the connections between these blocks.

The Simulink model was parameterized with full system parameters values and thereby manually entering the corresponding values of these parameters was avoided. A global file was defined that has to be loaded before using this model. The global file will load the values of these parameters so that Simulink will recognize the parameters. There are two main benefits for setting up the model in this way:

1. It makes changing and updating the values for the model easier, as the model obtains the values from only one source, namely the global file; and

2. It allows writing a MATLAB script that will perform the task of loading different values for these parameters and that will run the Simulink model multiple times. Also, each time the system runs, the system output is logged and stored separately for analysis and evaluation.

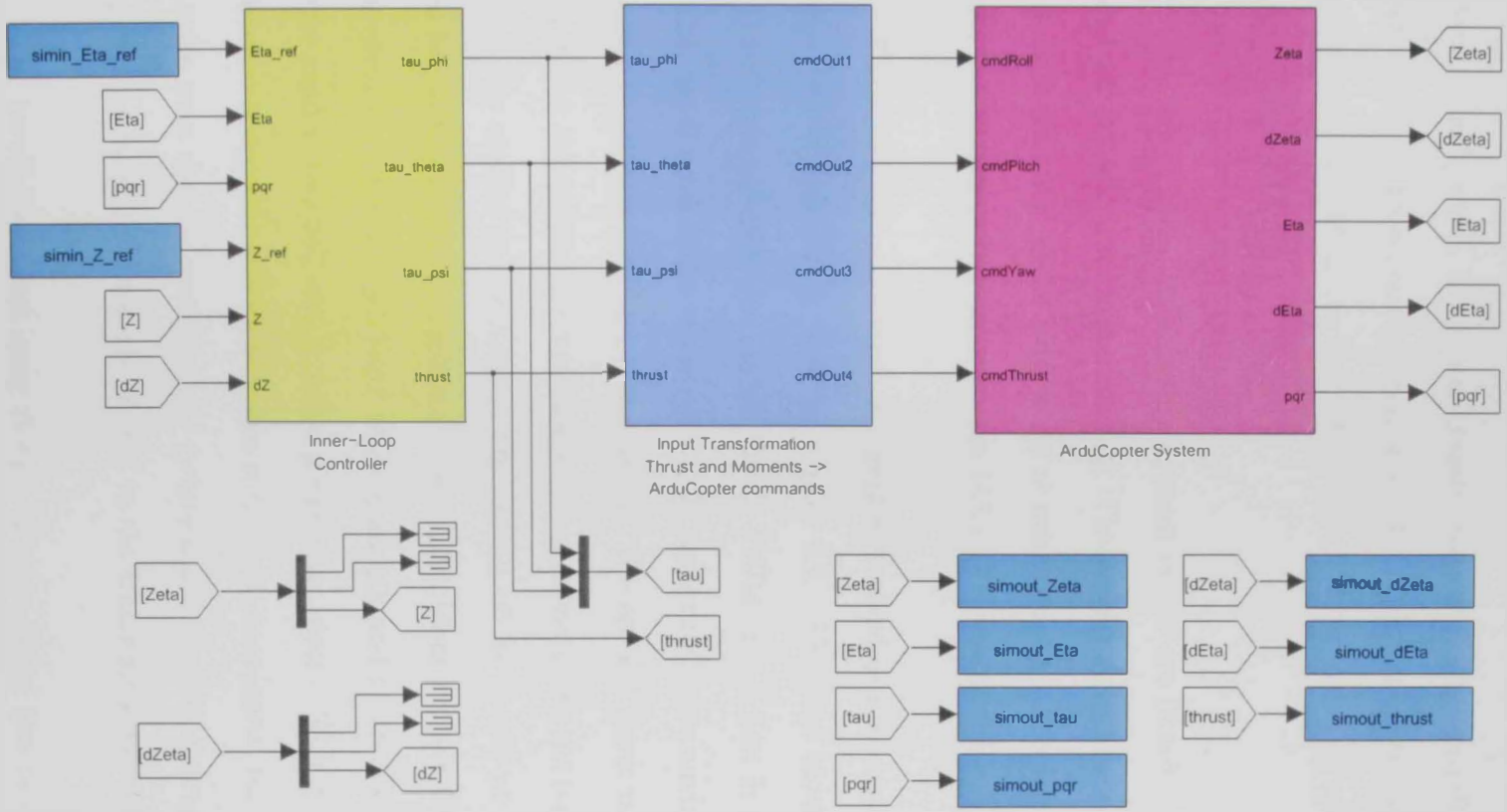


Figure 2.5: Simulink model of the quadrotor system

As can be seen in Figure 2.5, the complete quadrotor closed-loop system consists of the block "ArduCopter system", which is the plant, the block "inner-loop controller", which is the feedback controller, and the block "input transformations", which is a static transformation of the controller output, which was designed based on a simplified model. The controller block and the input transformation block are not relevant for this section and will be discussed in the Classical Control (Chapter 4).

The blocks other than the three main blocks are Simulink specific and not directly related to the system model. These supporting blocks are used to avoid using a complex model with many crossing signals while importing/exporting the simulation signals from/to the MATLAB workspace.

The ArduCopter platform as a complete quadrotor system consists of three main parts, as can be seen in Figure 2.6. The first block transforms the ArduCopter-specific commands into a PWM signal that is fed to the ESCs. This block models the software layer in the ArduCopter environment and captures the library used for this system. The second block models the motor-propeller subsystem as a dynamic system and produces the moments and forces that are applied to the frame. The output of the motor-propeller subsystem is fed to the block "quadrotor frame". This block models the airframe of the system as a flying rigid body with moments and forces affecting it. Finally, the model also includes a noise generation block. This block is controlled from a script file and its purpose is to produce moment noise and body rate noise with various specifications (white noise, maximum frequency, and noise power). These noise signals are fed to the frame for simulation purposes.

The break-down and inner design of these three blocks are discussed and explained in the following three subsections.

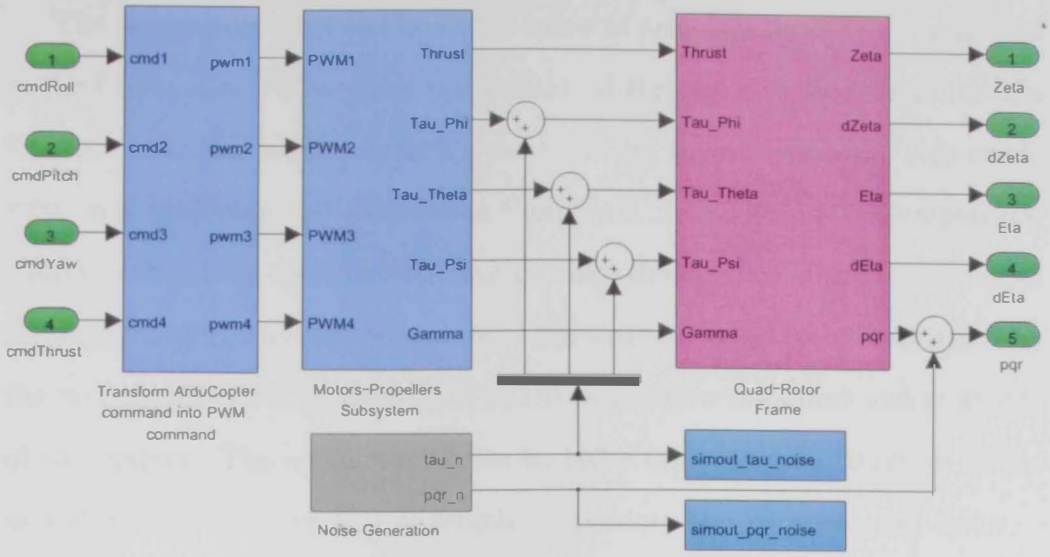


Figure 2.6: ArduCopter system model

2.6.2 Motor System

The motor-propeller subsystem is modeled as shown in Figure 2.7. This model captures the model of the ESC for the brushless motors as well as the dynamic response of the motor-ESC closed-loop system. This model also captures the forces and moments equations presented earlier in (2.10) and (2.11). These equations are combined into a single relation as follows:

$$U = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \\ T_t \end{bmatrix} = \begin{bmatrix} 0 & -bl & 0 & bl \\ bl & 0 & -bl & 0 \\ d & -d & d & -d \\ b & b & b & b \end{bmatrix} \begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} = M\hat{\Omega} \tag{2.19}$$

The matrix, M , is called the decoupling matrix as it decouples the actions of all the propellers into independent moments and a total thrust force. Writing the relations to be combined as (2.19) - eases deriving the coupling relation to obtain the propeller angular velocity from the desired moments and thrust. It also makes the model in Figure 2.7 more readable. The use of this transformation matrix will be presented for control purposes in the Classical Control (Chapter 4).

The model presented in Figure 2.7 takes as an input the PWM signals fed to the ESCs. The PWM signal is transformed through a static transformation function into the angular velocity produced by the subsystem. This static relation is explained and obtained in Chapter 3. A dynamical time response is added to this static transformation to produce the dynamic angular velocity of the four motors. Finally, the square of the angular velocities is multiplied by the decoupling matrix presented in (2.19) to produce the forces and moments of the system. The actuation matrix includes the physical properties of the propellers and the aircraft arm lengths to produce the moments and forces.

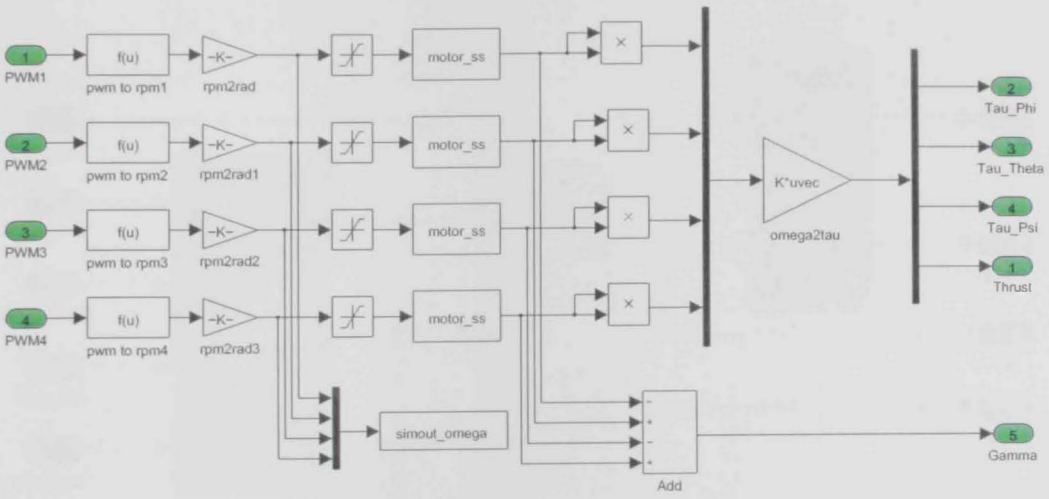


Figure 2.7: Motor-propulsion subsystem

2.6.3 Quadrotor Airframe

As can be seen in Figure 2.8, the quadrotor frame system consists of three main components. These three blocks are:

- The attitude dynamic model;
- The body-to-inertial frame of reference transformation; and
- The forces model.

The attitude dynamic model, colored in green, takes as an input the moments produced by the motor-propeller subsystem and builds up the angular

rates of the airframe represented in the body frame of reference. The attitude dynamic model captures and models the rotational motion of the airframe. The body angular rates are transformed from body frames of reference to derivatives of Euler angles as well as built-up Euler angles. This transformation is done in the frame of reference transformation block, colored in gray. Finally, the attitude of the airframe with the thrust applied to the airframe is fed to the forces model. Based on the thrust applied and the airframe attitude, the translational motion of the airframe is captured and modeled in the forces model, colored in green.

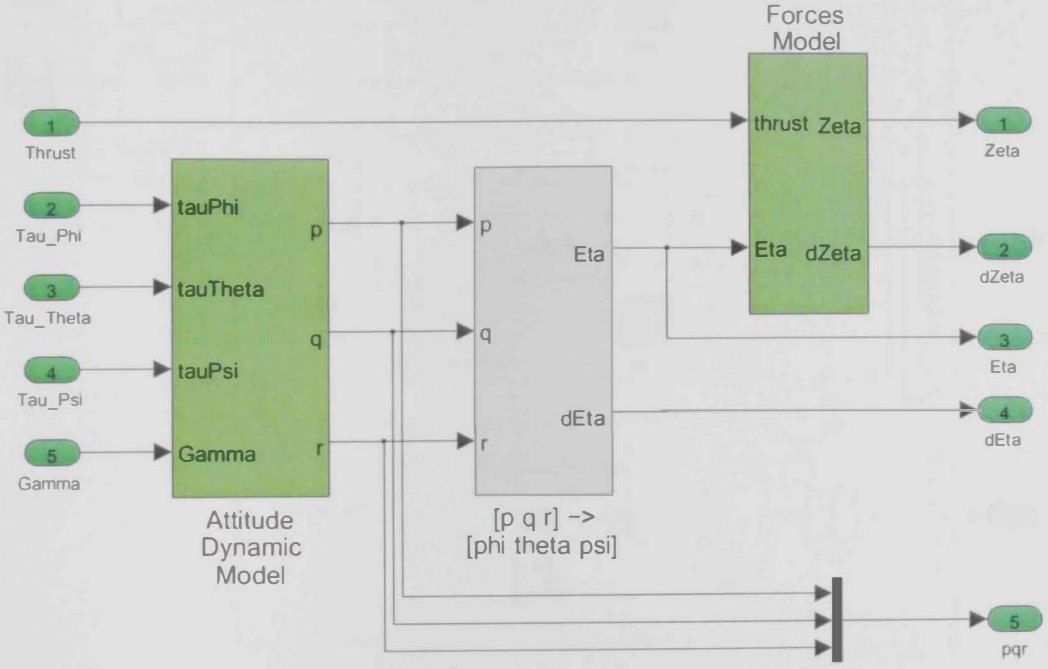


Figure 2.8: Simulink model of the quadrotor airframe

2.6.3.1 Attitude Model

The attitude dynamic model of the quadrotor system is built into Simulink as illustrated in Figure 2.9. This model takes the three moments as primary inputs and the computed value of γ and produces the corresponding angular rates of the quadrotor system on the body FoR. There are two sets of parameters that determine the behavior of this model:

1. The body inertias I_{xx}, I_{yy}, I_{zz}; and

2. The gyroscopic effect coefficient of propeller rotation, γ.

The attitude dynamic Simulink model also makes use of the global file parameters to read the values of these parameters. These parameters are embedded in the model as constant blocks (orange-colored blocks in Figure 2.9) with corresponding MATLAB variables as values. The parameters also have corresponding deviation variables that start with "d" (e.g., dIxx).

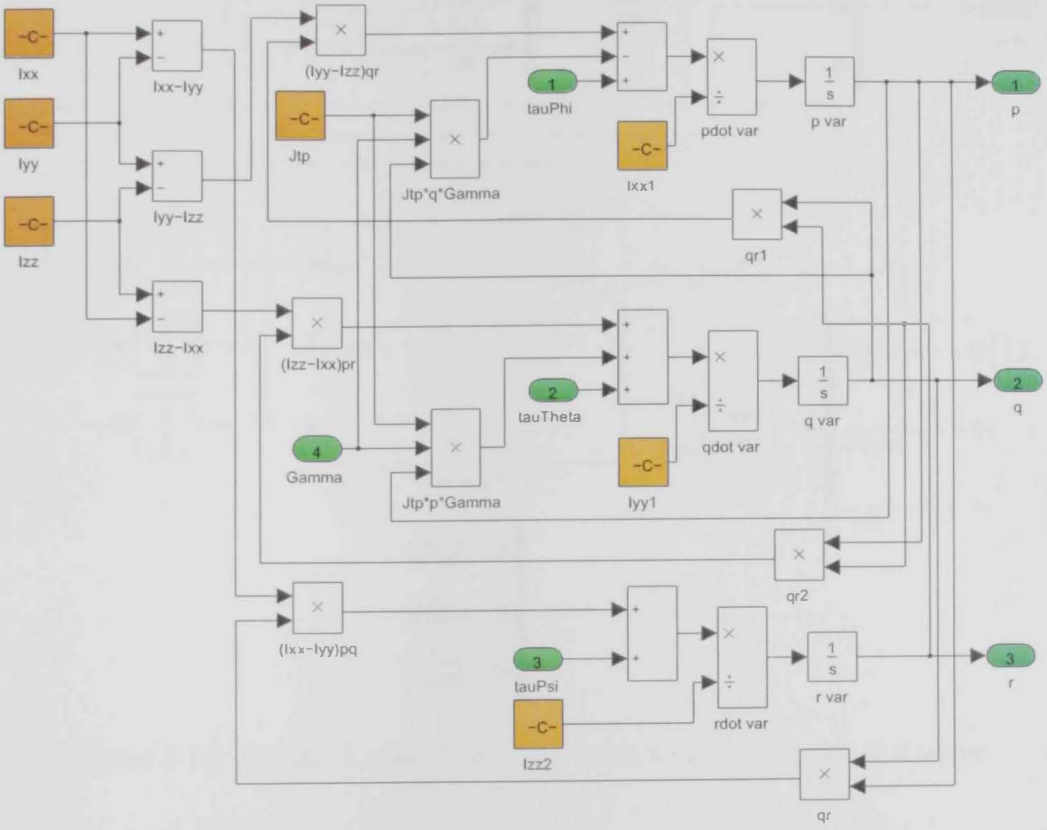


Figure 2.9: Attitude dynamic Simulink model

2.6.3.2 FoR Transformation

The produced angular rates of the quadrotor system with respect to body FoR are transformed and used to compute the Euler angles, ϕ, θ,and ψ. Figure 2.10 shows the model for obtaining Euler angles from the angular rates. The model takes the current angular rates, p, q and r, as inputs and produce as outputs the attitude vector, η, and its derivative, $\dot{\eta}$.

In this transformation block, there are three MATLAB variables needed to compute the output. These variables are the initial conditions for the three integrals. The three initial conditions are the initial state of the systems attitude. These initial conditions are set in the global file and they have a default value of zero.

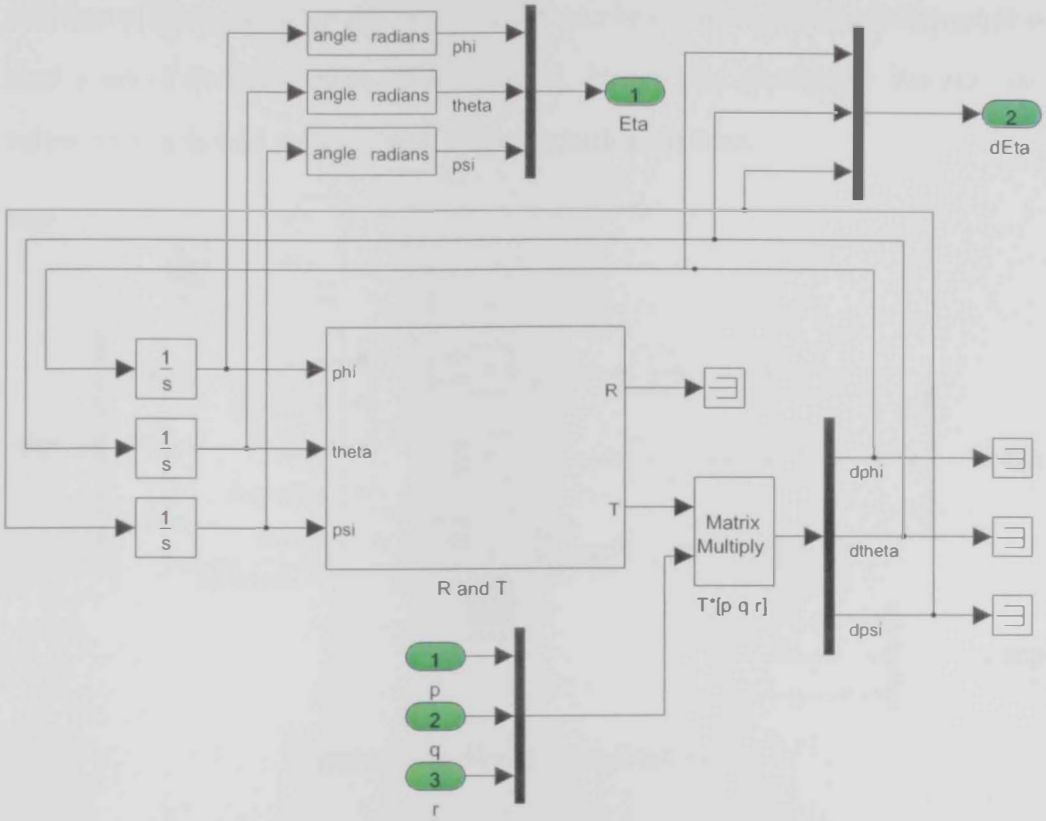


Figure 2.10: Simulink model of body rates to Euler angles and rates

2.6.3.3 Forces Model

The block diagram shown in Figure 2.11 is the realization of the mathematical model (2.14) in Simulink. The inputs to this block are the total thrust, T_t, and the attitude vector, η. At this level of the model, there are two parameters to be considered:

1. Mass, m; and

2. Gravity, g.

The gravity is modeled as a constant value. Although a more sophisticated model is possible,[1] it is not necessary and this simplification is sufficient for the purpose of this research. Just like the rest of the system parameters, these two parameters are included in the global file along with the corresponding deviation variable. This model uses double integration to produce the actual position of the quadrotor system from the accelerations and for each integration level a set of initial conditions is required. Hence, the global file also sets the values of the initial velocity and initial position vectors.

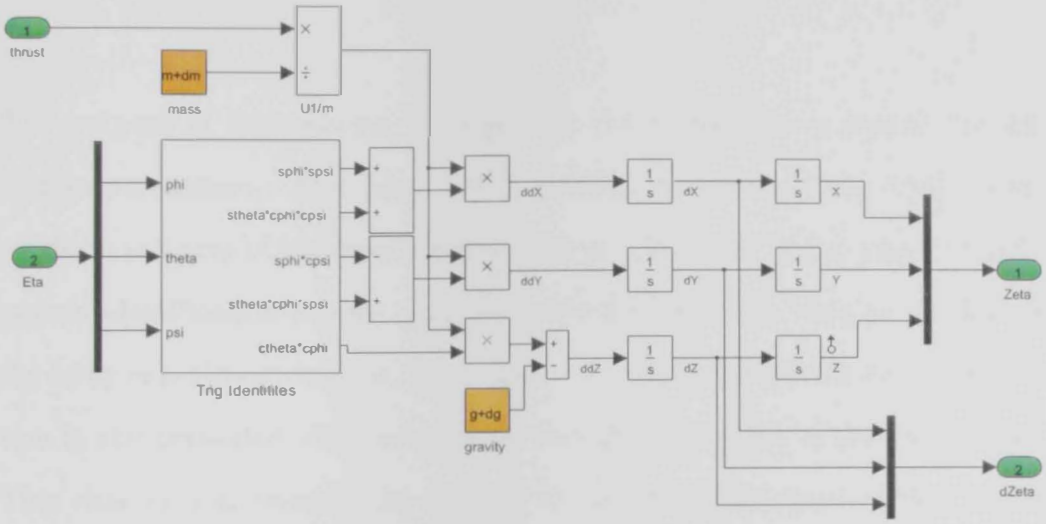


Figure 2.11: Forces Simulink model

[1] DTU10 is one of the available gravity models. The model gives more accurate estimate of gravity constant in different geographic areas (ie. close to mountains, oceans, and more.

Chapter 3

System Identification

The purpose of this chapter is to present the identification process for the system parameters. The chapter first gives an overview of the ArduCopter platform in terms of hardware decomposition. Then, the motor-propeller subsystem identification process is explained in details in this chapter, as well as the setup experiment used for this purpose. The data collected in the identification is also presented with the analysis and the estimation of the parameters. This chapter also presents the process of measuring the inertia using Miller pendulum-based method. Another identification step is also presented in this chapter to consolidate the results from the motor-propeller identification. Finally, this chapter presents the validation of the estimated parameters against collected data from experimental flights.

The objective of system (or platform) identification is to obtain the parameters that characterize the system. By looking at the model discussed earlier in Chapter 2, the parameters that need to be identified or measured are:

1. Propeller thrust and drag coefficients (b, d)
2. Platform mass (m); and
3. Platform inertias on three axes (I_{xx}, I_{yy}, I_{zz})

A few steps had to be taken to identify the system parameters. Initially, the motor-propeller system was taken independently to estimate the blade thrust coefficient as well as to determine at which angular velocity the motor driver will regulate for a given PWM command. This step was done off-line and focused on the motor-propeller mainly, without flying and allowing other factors coupling into the experimental data. The second step was to estimate the system inertias. For this purpose, a pendulum method was used. This method was proposed in the early 1900s for small-scale aircraft [24]. The third step was a step that was added to consolidate the estimated parameters in the identification process. This method is dependent on having flight data available and assumes that the inertia is estimated and the angular velocity of the system propellers is known. By deriving the moments from the system angular rates and mapping the propellers angular velocity with respect to the moments and torques, a linear relationship could be obtained and the blade thrust coefficient could be determined.

The third step was needed as the motor-propeller experiment had problems and vibrations in the setup, which made the results obtained questionable. Hence, the third method was used to verify the results.

3.1 ArduCopter Overview

As stated earlier, the platform chosen for this research was the 3DR ArduCopter. This is an open-source platform which is greatly supported by developers and hobbyists. The original frame is not the same as shown in Figure 3.1. The adapted frame is heavier and has more inertia than the original plastic frame. It is rugged and more resistant to damage in case of a crash or hard landing.[1] All the onboard software as well as the schematics of this system are readily available on the ArduCopters Wiki page. For these reasons mainly, this platform was chosen for our research.

Figure 3.1: The 3DR ArduCopter system

The ArduCopter system has a compact avionics kit that includes the following components:

- An Atmeg microcontroller unit (MCU);
- Triple-axis accelerometers;
- Triple-axis magnetometer;
- Triple-axis gyroscopes;
- A global positioning system (GPS); and
- A data flash.

[1] It did prove to be very resistant to damage in a couple of very hard crash landings that occurred during test flights.

The picture in Figure 3.2 shows the avionics kit of the ArduCopter.

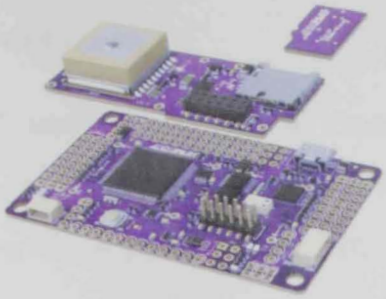


Figure 3.2: ArduPilot Mega Board v2 and its sensors

The system consists of six main components as shown in Figure 3.3, namely the:

- Avionics kit;
- Motor circuit drivers;
- Brushless motors;
- Propellers; and
- System airframe.

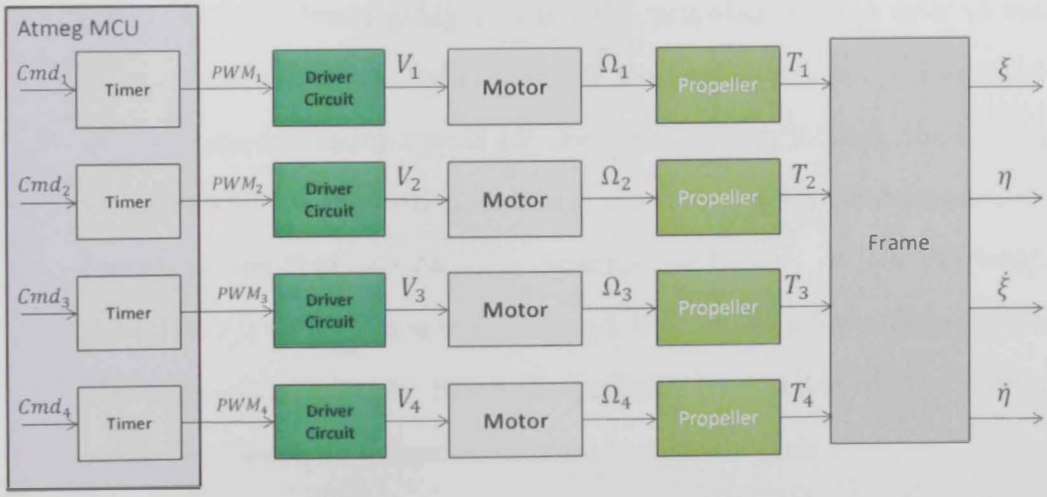


Figure 3.3: ArduCopter system breakdown

3.2 Motor-Propulsion System

3.2.1 Process Overview

The setup depicted in Figure 3.4 was used to identify the motor-propulsion system. As can be seen, this identification process was applied to the combination of motor and propulsion for simplicity. The objective of this process was to achieve mapping between the command fed to the motor driver and the thrust generated by the system. To achieve this objective, a simple setup was prepared from very basic components. This setup included the following three components:

1. Load-cell sensor:

   This sensor produces voltage at its output has a value proportional to the applied force of the sensor. See Figure 3.5 for the sensor used in this setup.

2. Laser pointer and phototransistor:

   These two components were used together to measure the angular velocity. The laser pointer is located on one side of the motor-propulsion setup, with its beam going through the propeller-covered area to the photo-transistor on the other side. If the voltage output is measured at the photo-transistor circuit (at the collector specifically), the output voltage should be high when the beam is not hitting the photo-transistor (which means that one blade is crossing the beam). When the beam goes through the blades area un-crossed, it produces a low voltage at the photo-transistor output. Since the propellers are made of two blades, every two measured pulses correspond to a single cycle.

3. Flexible mounting arm:

   This arm is attached to a bearing with very low friction.[1] At the end

[1] A bearing from a scrap hard drive was used for this application.

of this arm, the motor is mounted. This arm should be flexible to not reduce the thrust generated from the blades due to the friction in the arm joint and let this thrust transform into torque on this flexible arm. This torque will be measured as a force by the load-cell at another location on the arm.

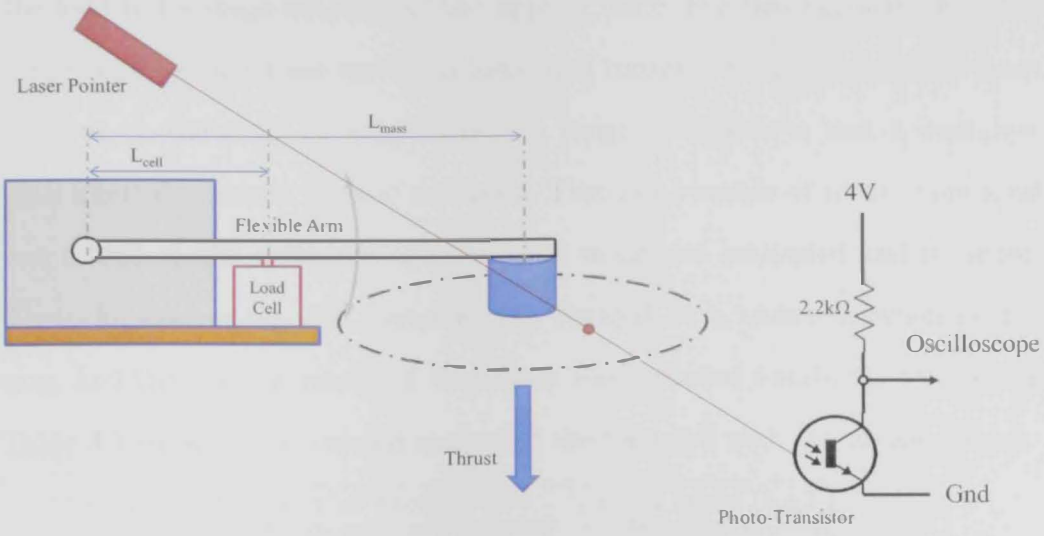


Figure 3.4: Motor-propulsion identification setup

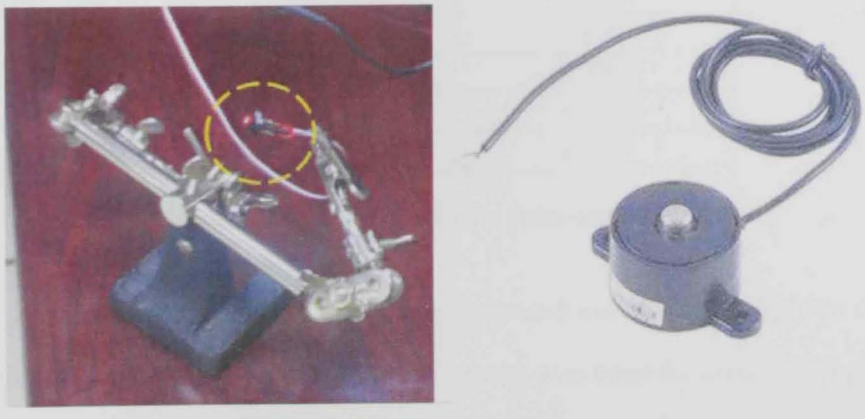


Figure 3.5: Photo-transistor and load-cell

A digital oscilloscope was used to gather three main signals from the setup:

- Load-cell voltage output;
- PWM command fed to the motor driver circuit; and

- Photo-transistor collector voltage (for angular velocity).

3.2.2 Force-Voltage Mapping

As a first step before conducting the experiment and running the motor, a calibration process had to take place to determine the relationship between the load-cell voltage output and the applied force. For this purpose, the same setup depicted in 3.4 was used, but instead of running the motor, a known mass was put on the arm. In order to have a range of masses to test, a container with a certain amount of sand was used. Through a couple of turns, more sand was added to the container and the total mass was measured and recorded. Then, for each mass, the container was hanged on a known location on the arm, and the voltage output of the sensor was recorded beside the total mass. Table 3.1 shows the measured output of the load cell with the applied mass:

Load Cell Output (V)	Loaded Mass (Kg)
0.924	0.213
1.066	0.304
1.205	0.394
1.355	0.481
1.555	0.607
1.698	0.697
1.833	0.785
1.996	0.895
2.254	1.054

Table 3.1: Load cell raw measurement

Transforming this data into a voltage-versus-force relationship was done in two steps. First, the weight (force) of the mass was transformed on its position to the load-cell position and then the appropriate line fitting was done.[1] For the first step, simple physics state that the weight will produce torque on the arm. This torque can be measured as a force at different positions on the arm, and the measured force will be a function of distance. The following is a simple derivation to obtain the applied force on the cell:

[1] According to the load cell manufacturer, the sensor has a linear output to the applied force.

$$\tau_{mass} = F_{mass} L_{mass} = F_{cell} L_{cell}$$

$$F_{cell} = \frac{m_{mass} g L_{mass}}{L_{cell}}$$

For the setup shown, the following parameters where measured for the previous relationship:

$$L_{mass} = 0.113m$$

$$L_{cell} = 0.08m$$

After applying the previous transformation, the mapping between the sensor voltage output and applied force could be obtained using the data shown below in Table 3.2 and Figure 3.6.

Voltage (v)	Force (N)
0.924	2.9515
1.066	4.2124
1.205	5.4595
1.355	6.6650
1.555	8.4110
1.698	9.6581
1.833	10.8775
1.996	12.4017
2.254	14.6049

Table 3.2: Load cell force-voltage relationship data

The relationship between the voltage output of the load cell and the applied force is linear as can be seen in Figure 3.6. A simple line equation with a slope and bias was deemed to be sufficient. Using the "basic fitting" function in MATLAB, the following relationship was obtained:

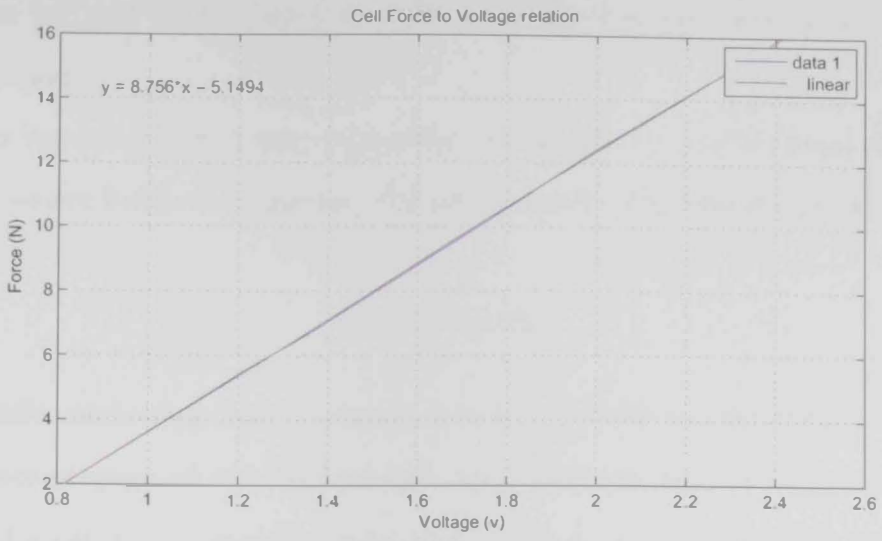


Figure 3.6: Load cell force-voltage relationship

$$F(v) = 8.756v - 5.1494$$

It is important to note that the bias term comes from two sources. One is the sensor voltage bias at no load and another is the load from the setup designed, as the arm and all components are applying an added force on the sensor. As the relationship is linear between the applied force and the measure voltage, the combined bias term (including sensor bias and the load from the setup designed) can be ignored in the transformation. As for the main identification experiment, the bias had to be measured and removed from the data. This means that this relationship would only provide the added force produced by the propulsion system from the no-load case, which was the main purpose of this exercise.

3.2.3 Experiment

At this point, the identification experiment could be conducted. The procedure followed in this experiment was simply to give a range of commands to drive the motor at a range of fixed angular velocities to construct a table, Table 3.3, consisting of command (PWM), the root mean square (RMS) of load cell

voltage (v), and the frequency of pulses resulting from the blades crossing the laser beam.

An important step before conducting the experiment was to record the bias of the sensor before commanding the motor driver. The bias was found to be:

$$Bias = 0.643V$$

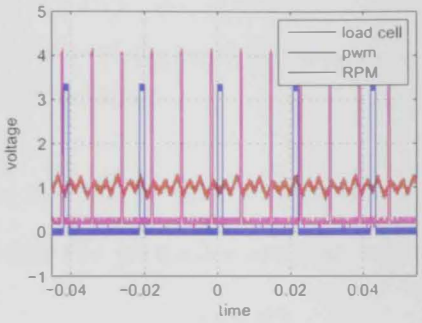
While conducting the experiment, excessive vibrations existed in the setup and were propagated into the load cell. This vibration was noticed to resonate around a certain command, which corresponds to a certain angular velocity, as can be seen from the plot in Figure 3.9a.

The source of vibration was identified to be from the motor and the propeller, combined. This can be noticed in the oscilloscope figures, Figures 3.7a, 3.7b and 3.7c. The fact that the vibration disturbing the load-cell output was very well synchronized with the angular velocity signal supports this argument, keeping in mind that every pulse on the angular velocity signal corresponds with a single blade. Some effort was made to enhance the setup and remove the vibration, but this enhancement was not sufficient. As this research had to focus on studying robust control, this issue with the vibration was left for a subsequent study. Nevertheless, the noisy load-cell output was handled by taking the RMS value from the oscilloscope. Table 4.3 summarizes the recorded measurements and calculations by the oscilloscope for voltage RMS and angular rate:

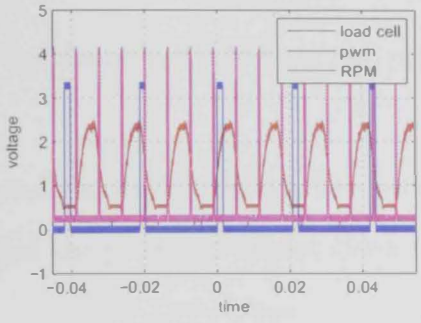
The function $F(v)$ obtained from the previous step was then used to calculate the force measured at the load cell. Similarly to the force-voltage mapping step, this force should be transformed from the load-cell position point to the motor position point on the arm. In this case, $L_{motor} = 0.2670m$ and the load cell is at the same position. Moreover, a conversion had to be done for the frequency of the pulses produced from the blades crossing the laser beam (the "frequency" field in the table). This conversion considers the fact that the

Command PWM (msec)	Load Cell RMS (v)	angular rate (Hz)
1.2	0.725	59.73
1.298	0.867	96.28
1.39	1.05	122.5
1.499	1.18	144.2
1.595	1.49	159.2
1.705	1.59	182.3
1.748	1.65	191.5
1.800	1.79	204.9
1.845	1.88	215.9
1.917	2.10	230.1

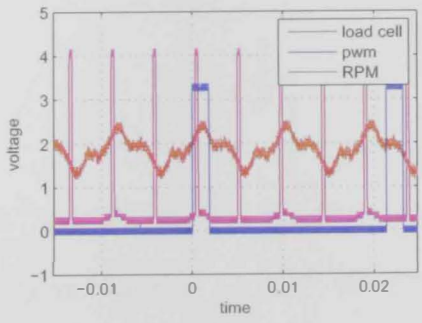
Table 3.3: Motor-propulsion identification data



(a) Low command



(b) Command at maximum vibration



(c) High command

Figure 3.7: Signals captured from oscilloscope during identification test

propeller has two blades and converts the frequency of the propeller pulses into angular velocity. This conversion is done as follows:

$$RPM_{prop} = \frac{Frequency}{2} \times 60$$

Finally, the figures for the purpose of this identification exercise could be obtained. Figure 3.8 shows the relationship between the command (in PWM

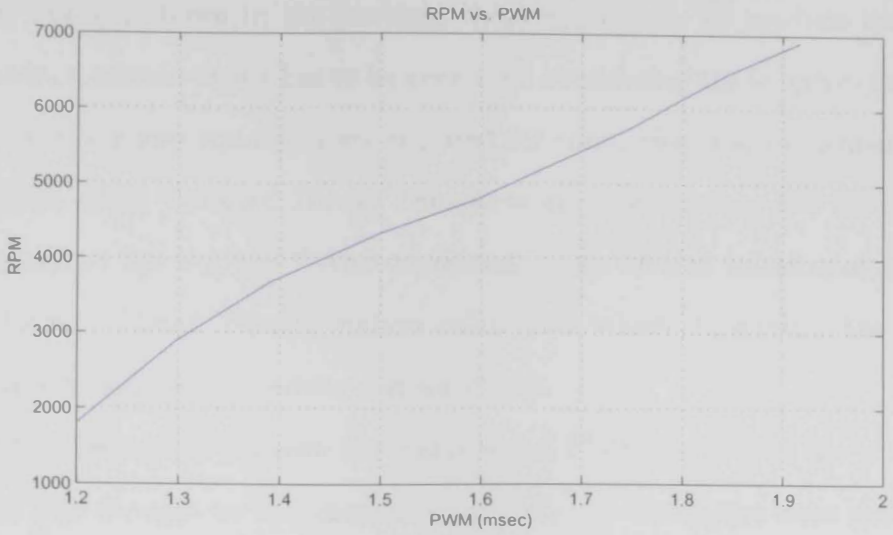
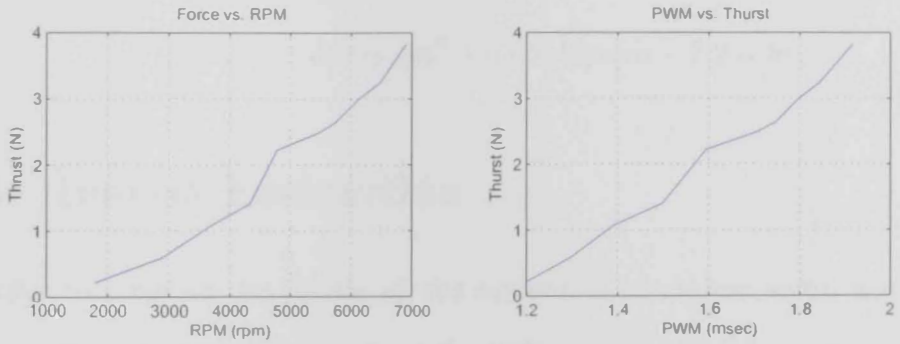


Figure 3.8: Relationship between motor command in PWM form and propeller angular velocity

form) and the propeller angular velocity and Figure 3.9a shows the relationship between the propeller angular velocity and the produced thrust (force).



(a) angular velocity vs Thrust.

(b) PWM vs Thrust.

Figure 3.9: Relationship plots between PWM, angular velocity, and thrust.

Out of the three figures, two are important for this research. Figure 3.9a shows the relationship between the propeller angular velocity and the produced thrust. This relationship characterizes the propeller used and can be included into the model of the system. Figure 3.9b is important for the control system implementation. As the designed controller produces the control signal in units of moments and thrust, it was critical to convert the control signal from this form into required PWM signals that will feed into the real system in flight

tests. As was shown in the previous chapter, in order to produce a certain moment, a certain thrust has to be generated considering the length of the arm. The following two equations are required for simulation and implementation. Equation (3.1), *Forward Thrust Transformation*, determines the thrust that is generated for a given PWM command. The inverse relation of (3.1) is the *Inverse Thrust Transformation* calculation which determines the PWM command required to produce a given thrust.

The relationship between the commanded PWM signal and the produced thrust was thought to be exponential initially. However, the data show that a quadratic regression technique will be more conducive to fitting the data in this case. Using the "basic fitting" function in MATLAB, the quadratic relationship was obtained to calculate the produced thrust for a given PWM command, as depicted in (3.1).

$$T(pwm) = 1.3488pwm^2 + 0.66218pwm - 2.25108 \tag{3.1}$$

3.3 Inertia Estimation

In order to measure the inertia of the system, a pendulum setup was used. This method was suggested in the early 1930 by Miller [24] for measuring the inertia of small-scale aircrafts. Miller proposed two setups. One setup, on the left-hand side in Figure 3.10, is used to measure inertia about the z-axis by swinging the aircraft about the vertical axis. The other setup, on the right-hand side in Figure 3.10, is used to measure inertia about the horizontal x-axis by swinging the aircraft about the axis of oscillation at the top as in the figure.

In this research, due to time constrains, a simple, single setup was implemented to measure all inertias about the three axes. The quadrotor was oriented in order to measure the inertia for the axis of interest. This setup is shown in Figure 3.11 with the two parameters indicated. The experiment was conducted by swinging the quadrotor about the vertical axis for any desired

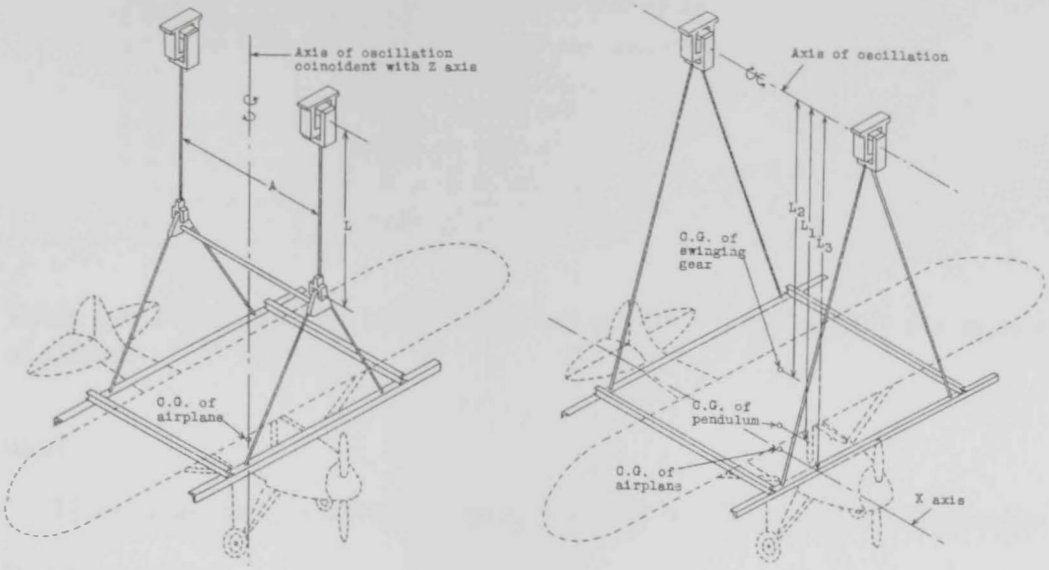


Figure 3.10: The two setups suggested by [24] to measure I_{xx} and I_{zz}.

axis.

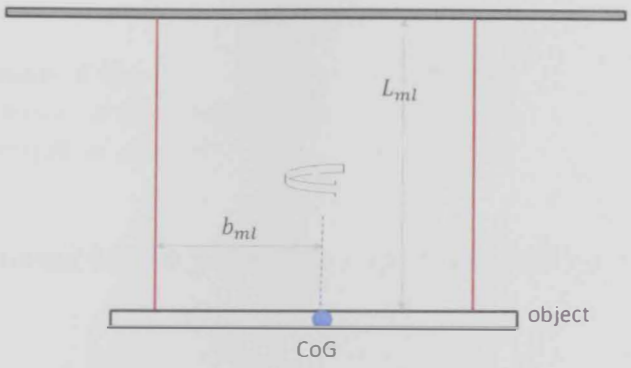


Figure 3.11: Inertia measurement setup

The equation that governs this experiment is:

$$I_{ml} = \frac{W_{ml} T_{ml}^2 b_{ml}^2}{4\pi^2 L_{ml}} \tag{3.2}$$

The parameters in Miller setup equation (3.2) are listed in Table 3.4.

Initially, an object with a uniform shape was selected for inertia measurement. A uniform object was needed to be able to calculate its inertia theoretically, using the standard equations for uniform objects and compare the theoretical result with that of the experiment. For this first step a rod was

Parameter	Definition	Units
I_{ml}	: Inertia of aircraft	$kg.m^2$
T_{ml}	: Period of oscillation	$seconds$
W_{ml}	: Weight of aircraft	N
b_{ml}	: Radius of oscillation ($b_{ml} = D/2$)	m
L_{ml}	: Length of vertical lines	m

Table 3.4: Parameters for Miller setup to experimentally calculate the inertia of aircraft.

used.

The rod inertia was calculated using the solid cylinder equation. The equation is as follows:

$$I_{rod} = \frac{M_{rod}R_{rod}^2}{4} + \frac{M_{rod}L_{rod}^2}{12}$$

Where:

M_{rod}	is the mass of the rod	= 0.093 kg
R_{rod}	is the radius of the cross-section	= 0.011 m
L_{rod}	is the length of the rod	= 1.200 m

After substituting for the values, the rods inertia will be:

$$I_{calc} = \frac{MR^2}{4} + \frac{ML^2}{12}$$
$$= 11.163g.m^2$$

After obtaining the inertia of the rod theoretically, the rod inertia was measured experimentally. When conducting the experiment, the rod was rotated about 10^o and then it was released to oscillate. The rod was left to oscillate for 10 cycles and then the average period was taken. The experiment was conducted with the following parameters and results:

The measured inertia was:

$$
\begin{aligned}
T &= \tfrac{38.9}{10} sec = 3.89 sec \\
W &= M_{rod} \times g \\
b &= 0.20m \\
L &= 1.27m
\end{aligned}
$$

$$
\begin{aligned}
I_{meas} &= \frac{0.093 \times 9.81 \times 3.89^2 \times 0.2^2}{4 \times \pi^2 \times 1.27} \\
&= 11.014 g.m^2
\end{aligned}
$$

The absolute difference between the theoretical and the experimental results is $0.149 g.m^2$. The relative difference between the two results ($11.163 g.m^2$ and $11.014 g.m^2$) is approximately 10%. This difference is considered acceptable. In order to gain further confidence, another step was included in the experiment. In this step, any uncertainty from the theoretical calculation was reduced, considering that for the formula used to calculate the inertia theoretically it had been assumed that the rod is a perfect cylinder, which was not the case. In order to avoid this ambiguity in the calculation, this additional step was included to remove the ambiguity of the theoretically calculated inertia.

Generally speaking, for any object with unknown inertia, adding a known mass at a certain position from the CG will add a certain amount of inertia to the inertia of the original object. This increase in inertia is calculated with the following basic formula:

$$I_\delta = ml^2$$

For the experiment in this study, two known masses of $0.44kg$ each were added to the rod at $0.53m$ on each side of the rod to keep the CG unchanged. This addition of the two masses should increase the inertia in comparison with what it was in the previous experiment by:

$$I_\delta = 2 \times (ml^2)$$
$$= 2 \times (0.440.53^2)$$
$$= 167.28g.m^2$$

When conducting the experiment again with the added masses and keeping all the other parameters unchanged, the new period of oscillation was $4.61sec$, which gave an inertia of:

$$I_{meas} = \frac{(0.093 + 2 \times 0.44) \times 9.81 \times 4.61^2 \times 0.2^2}{4 \times \pi^2 \times 1.27}$$
$$= 177.185g.m^2$$

The difference between the new measurement and the previous measurement was:

$$I_\Delta = 177.185 - 11.014$$
$$= 166.171g.m^2$$

The absolute difference in the delta step was $I_\delta - I_\Delta = 1.109g.m^2$. If this difference is considered on the scale of $\approx 166g.m^2$, it is assumed to be very acceptable (less than 1%).

At this stage, after all the analyses and experiments have been completed, the setup depicted earlier was deemed to be sufficient and give enough confidence to this experiment to obtain the inertias of the quadrotor platform.

Using the same setup shown earlier in Figure 3.11, the quadrotor was hanged on a simple pendulum setup as shown in Figure 3.12. The experiment was conducted separately for each axis. The inertia I_{xx} was measured first and then I_{zz}. I_{yy} was assumed to be equal to I_{xx} which is a very valid

assumption for symmetrical platforms such as quadrotors.

Figure 3.12: Quad-rotor in the Miller setup

For each axis experiment, three trials were attempted with 10 oscillations in each trial. The 10 oscillations were averaged individually for each trial and then the total average of all the trials was taken. Table 3.5 shows all the recorded experiment data:

Trial No.	T records for I_{xx} (sec)	T records for I_{zz} (sec)
1	0.85	1.35
2	0.84	1.35
3	0.85	1.36
Average	0.847	1.353

Table 3.5: Records of the inertia measurement experiments for I_{xx} and I_{zz}.

The experiment parameters for the quadrotor setup were as follows:

$W = (1.153kg) \times g$
$b = 0.25m$
$L_{Ixx} = 0.82m$
$L_{Izz} = 0.83m$

Using the experiment equation (3.2), the inertia values could be calculated as in the previous steps.

$$I_{xx} = 15.654 \times 10^{-3} kg.m^2$$

$$I_{zz} = 39.514 \times 10^{-3} kg.m^2$$

Further validation will be done in the next section by comparing the outputs of the simulation model against preliminary data collected from experimental identification flights.

3.4 Backward Identification

The purpose of backward identification method was to estimate the blade coefficient from experimental flights. This step would add more confidence to and consolidate the results obtained earlier. In this method, the results obtained from the motor/propeller thrust setup were ignored. The only outcome of the motor-propeller setup considered was theangular velocity-PWM mapping to obtain the motors angular velocity from the logged PWM commands. This step also required the inertia of the system to be known with a certain degree of confidence, as well as the distance between the motors centers and the quadrotor's CG.

Starting with the common moment relation:

$$\tau = I.\alpha$$

Where α is the angular acceleration, and I is the inertia. If the inertia is

known, the torque input in flight, $\hat{\tau}$, that excited the system can be calculated for the three axes. The torque, $\hat{\tau}$, is the backward estimated torque from the derivative of the system bodys angular rates. Therefore:[1]

$$\begin{bmatrix} \hat{\tau}_\phi \\ \hat{\tau}_\theta \\ \hat{\tau}_\psi \end{bmatrix} = \begin{bmatrix} I_{xx} \\ I_{yy} \\ I_{zz} \end{bmatrix} \circ \begin{bmatrix} \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} \tag{3.3}$$

From (2.11), it is known that:

$$\begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \end{bmatrix} = \begin{bmatrix} bl(\Omega_4^2 - \Omega_2^2) \\ bl(\Omega_1^2 - \Omega_3^2) \\ d(\Omega_1^2 + \Omega_3^2 - \Omega_2^2 - \Omega_4^2) \end{bmatrix} \tag{3.4}$$

From these relationships, it can be seen that the blade coefficient and the drag coefficient are proportional to the torque. If the following functions are considered:

$$f(\Omega) = \begin{bmatrix} f_\phi(\Omega) \\ f_\theta(\Omega) \\ f_\psi(\Omega) \end{bmatrix} = \begin{bmatrix} l(\Omega_4^2 - \Omega_2^2) \\ l(\Omega_1^2 - \Omega_3^2) \\ (\Omega_1^2 + \Omega_3^2 - \Omega_2^2 - \Omega_4^2) \end{bmatrix} \tag{3.5}$$

Then the blade thrust coefficient and drag coefficient can be calculated by mapping $\hat{\tau}$ versus $f(\Omega)$, as the relation between both is:

$$\hat{\tau} = \begin{bmatrix} b \\ b \\ d \end{bmatrix} \circ f(\Omega) \tag{3.6}$$

It is important to note that when using the static transformation obtained before to calculate the angular velocity (Ω) from the logged PWM signal, a dynamic response of unity gain has to be considered and applied to the Ω signal before mapping. Further, the dynamic response should have the same

[1] The symbol $\circ$ is for Hadamard product (element-wise product)

time constant, τ_{mp}, as the motor-propulsion system. The dynamic response is a simple unity gain, first-order system with a 0.1 seconds time constant:

$$H_{mp} = \frac{1}{\tau_{mp}s + 1} \tag{3.7}$$

In Figure 3.13, a plot of the estimated torque and the actual torque is shown for the two different cases one with static transformation of PWM into angular velocity and the other with added dynamic response. It can clearly be seen that adding the dynamic response is required in order to obtain good estimates for the coefficients.[1] In the physical sense, the motor applies a certain torque for a certain period of time and the motor angular velocity takes a certain time to reach the steady-state angular velocity.

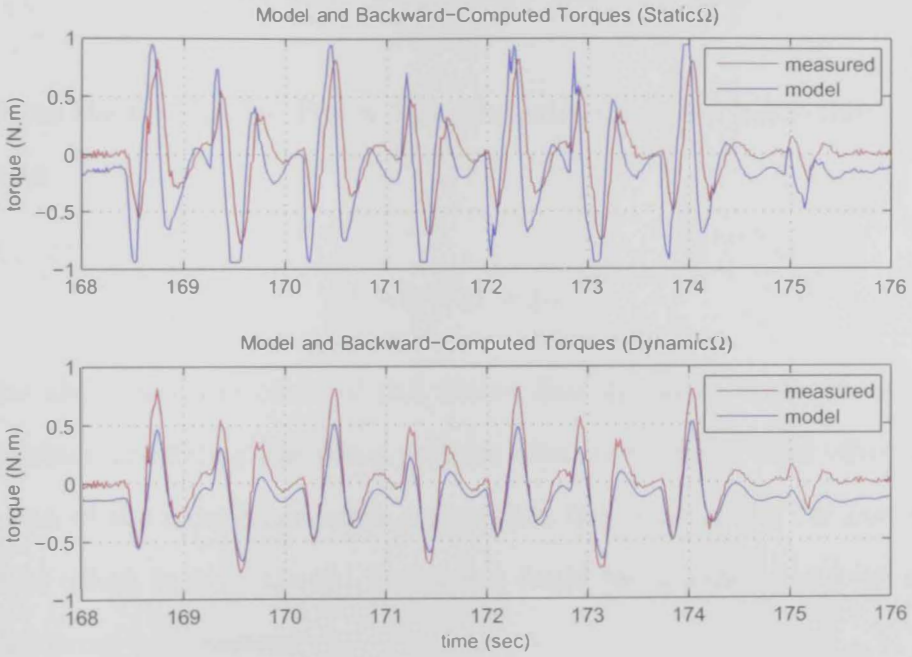


Figure 3.13: Torque comparison for static and dynamic Ω

The mapping of $\hat{\tau}$ versus $f_\phi(\Omega) = l(\Omega_1^2 - \Omega_3^2)$ is shown in Figure 3.14. Using simple line fitting, the slope of the fitting line can be calculated and the result should be the estimated value of blade thrust coefficient, b.

[1] The plots shown in Figure 3.13 are for the thrust computed in the motor-propeller setup. These results are used only to show the importance of applying the dynamic response.

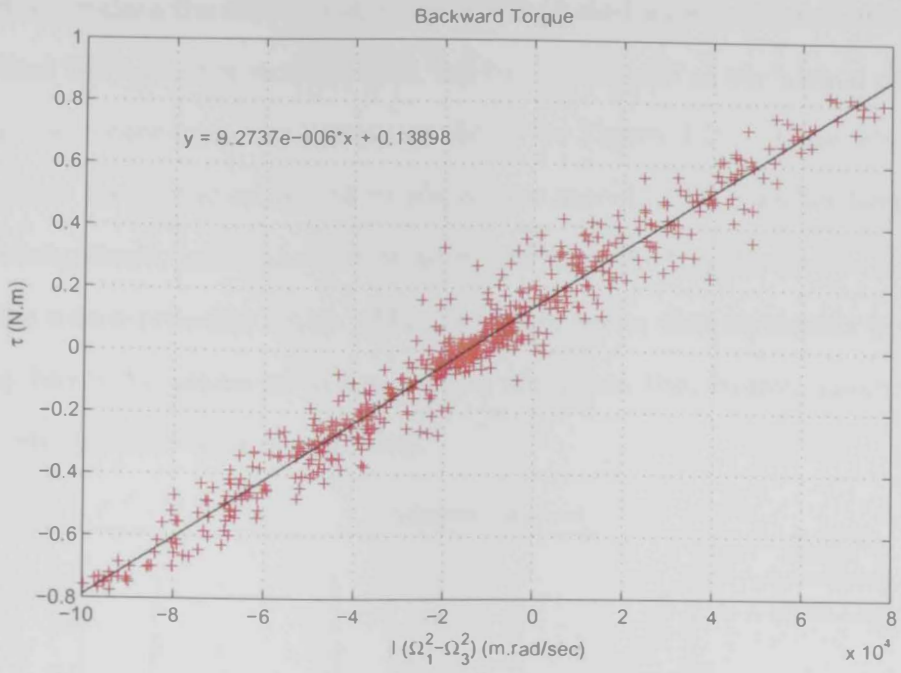


Figure 3.14: Mapping of $\hat{\tau}$ and $l(\Omega_1^2 - \Omega_3^2)$

From the mapping in Figure 3.14, the estimate of the blade-thrust coefficient is:

$$b = 9.2737 \times 10^{-6} \tag{3.8}$$

In the ideal case, the offset of this fitting line should be zero. However, in our system, the fitting line has a positive offset of 0.13898. This offset means that one of the motors has more action than the other motor (in the case of positive offset, it is motor-1). This offset could be caused by at least one of the following two reasons:

1. A shift in CG towards motor-1, which means that the subsystem motor-propeller-1 should have more action to compensate for this shift in CG.

2. The subsystem motor-propeller-1 is less efficient than the sub-system motor-propeller-3, which also means that Ω_1 will have to be higher so that sub-system-1 will have equal action to sub-system-3.

Finally, as an initial validation, the estimated blade thrust coefficient was

used to produce the roll torque, τ_ϕ, and it is plotted with the estimated torque, $\hat{\tau}_\phi$, that the system is excited with. All torques as well as the torque obtained from the motor-propeller setup are shown in Figure 3.15. It can clearly be seen that the torque estimated by the torque model (2.11) matches better at a high magnitude, unlike the torque estimated from the transformation obtained by the motor-propeller setup. This fact emphasizes that the motor-propeller setup has to be enhanced to remove the vibrations that caused inaccuracy in the results at the high magnitudes.

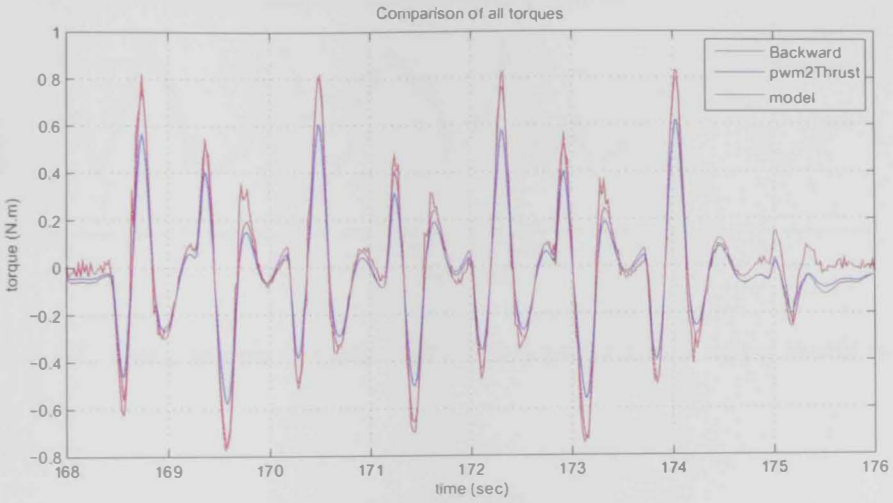


Figure 3.15: Comparison of all torques that are estimated in identification process.

3.4.1 Yaw Model Mismatch

When applying the backward identification procedure to estimate the drag coefficient, d, a suitable estimate was much more difficult to achieve. This was due to two main reasons. The main cause of this problem, was the fact that the yaw-torque, τ_ψ, was modeled as the drag from the propellers angular speeds (as in the model (2.11)). In Figure 3.16, two plots are shown, one for static $f_\psi(\Omega)$ and the other for dynamic $f_\psi(\Omega)$, with an applied time-response unity gain transfer function. It can be seen that the plot with static transformation is more suitable (but not as accurate as for the blade-thrust coefficient) to

estimate the drag coefficient, d. Figure 3.17 shows the relation between $\hat{\tau}_\psi$ and static $f_\psi(\Omega)$.

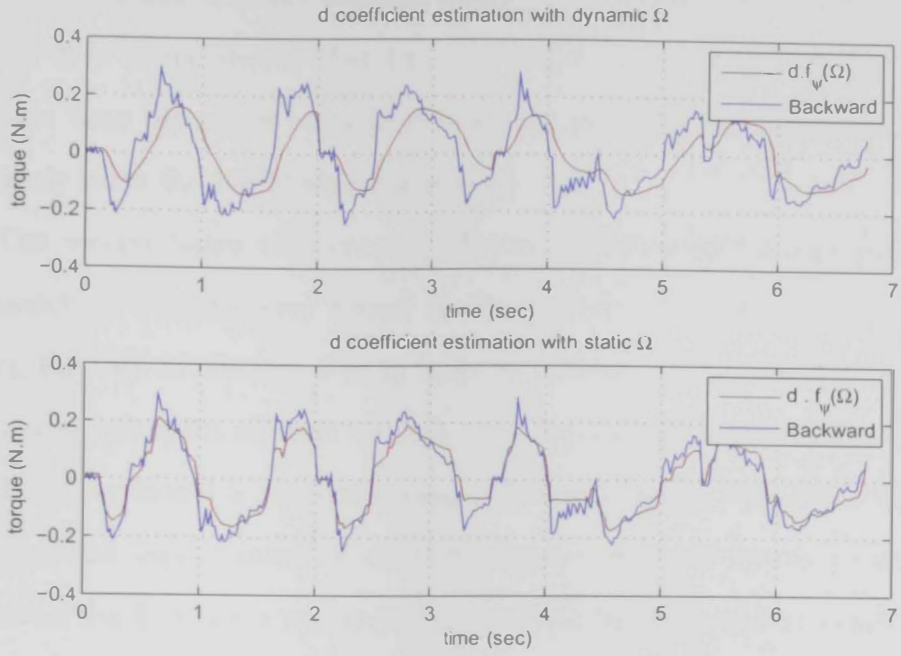


Figure 3.16: Comparison between estimated yaw-torque using static and dynamic Ω

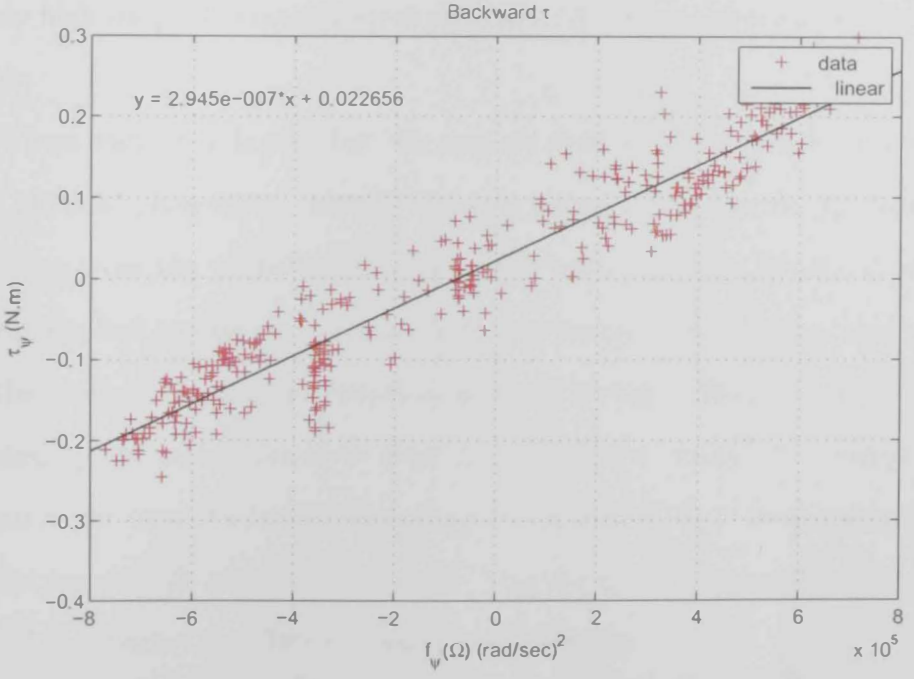


Figure 3.17: Mapping of $\hat{\tau}_\psi$ and $f_\psi(\Omega)$ using static Ω.

Considering $f_\psi(\Omega)$ is obtained by static transformation of a PWM signal

into an angular velocity signal, it makes sense to apply a dynamic response to this signal as the motor needs a time constant to reach a certain angular velocity. The fact that the backward identification does not give good results with a slow signal means that the yaw-torque, $\hat{\tau}_\psi$, that excites the system is faster than the motor angular velocity signal, $f_\psi(\Omega)$, as it happens almost instantly when the PWM signal is applied.[1]

The second factor that caused difficulty in estimating d was the model mismatch of the generated torque in the system. Formula (2.13) for calculating the rate of change of body angular rate about the z-axis, $\dot{r}$, includes the gyroscopic effect of body rotation in 3D space and torque applied on the z-axis. If the model is accurate enough, then an impulse torque should result in a constant angular rotation, since the torque causes an angular acceleration. However, the flight data in Figure 3.18 indicate that the system maintained a constant torque in order to maintain the constant angular rate. The system maintained about 25% of the initial torque as the body angular rate reached about $2rad/sec$. It is believed that the system maintained this much of relatively high torque to compensate for the drag that resulted from body rotation in air.

These factors indicate that the system torque on the z-axis comes from two physical phenomena, namely the anti-torque as the dominant source and the drag from the propellers rotation. The anti-torque on the frame (due to motor-applied torque on the propeller) is believed to be the dominant factor in the systems angular acceleration along the z-axis. Estimating the torque applied by the motors could be done in a subsequent study. The torque of the motor in the case of a brushless motor is a function of the three phase currents, as discussed in [2] and [36] (i_a, i_b, i_c). Therefore, the suggested z-axis torque model is expected to relate to these three currents.

It is impossible to carry out this step of identifying the motor torque as

[1]The yaw-torque model should include motor anti-torque as well as propeller drag.

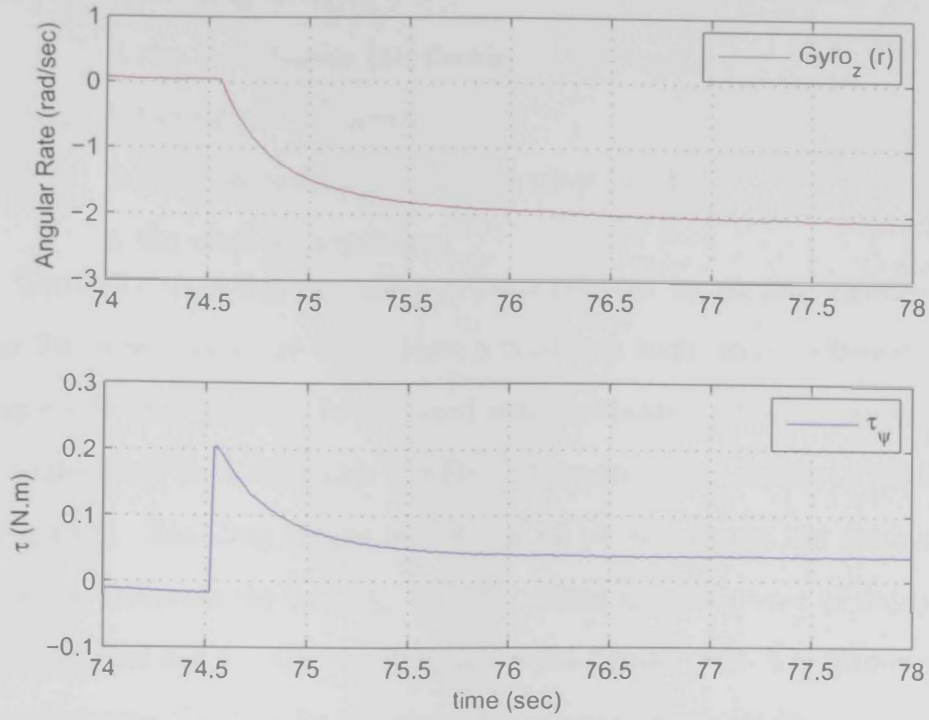


Figure 3.18: Sample of flight data for angular rate on the z-axis

the torque is closely related to the current consumed by the motors. The measurements of the individual motors current consumption in flight is not available for the ArduCopter platform. Adding this capability to the system is time and effort consuming and it was considered unnecessary for the main purpose of this research.

Another suggested improvement to the system model is to expand the model for the body angular rate about the z-axis. The new model adds the drag of the system rotation in air to the previous model. The general model of the drag force for an object's motion in air is - obtained from [32] -:

$$F_D = \frac{1}{2}\rho v_{as}^2 C_D A \tag{3.9}$$

Where:

F_D is the drag force;

ρ is the fluid density (air density in this case);

v_{as} is the aircraft air speed;

C_D is the drag coefficient (dimensionless); and

A is the cross sectional area.

Generally speaking, flat surfaces perpendicular to air flow (which is the case for most quadrotor arms) have a relatively high drag coefficient.[1] The drag coefficient, C_D, can be obtained using software tools that are based on computational fluid dynamics (CFD) and translational motion is calculated using (3.9). The drag torque model can be obtained from the translational model by replacing the term v_{as} with Lr, where L is the center of drag due to the rotational motion, and r is the angular velocity for yaw. The final equation for calculating the drag for the aircraft's rotation in the air is:

$$\tau_D = \frac{1}{2}\rho L^2 C_D A r^2 \tag{3.10}$$

For practical purposes, another model can be obtained from the relation in (3.11) by combining the terms ρ, C_D, A and L into a single parameter, λ, as in (3.12). The parameter λ can be obtained experimentally in the case of standard air density.

The final model of the aircraft rotation about the z-axis is:

$$\dot{r} = \frac{I_{xx} - I_{yy}}{I_{zz}} pq + \frac{\tau_\psi}{I_{zz}} - \frac{\lambda}{I_{zz}} r^2 \tag{3.11}$$

Where:

$$\lambda = \frac{1}{2}\rho C_D A L \tag{3.12}$$

[1]This is could be a guideline to enhance the aerodynamics of a quadrotor's airframe.

3.5 Validation

In order to identify the estimated plant parameters, the logged data obtained from the experimental flight were used and compared with the output of the system model in Simulink. The simulated model was injected with the same inputs that were logged from the flight and the output was plotted together with the actual system output of the flight. In Figure 3.19 the simulated system model output for body roll rate is shown in yellow and the actual system body roll rate is shown in pink.

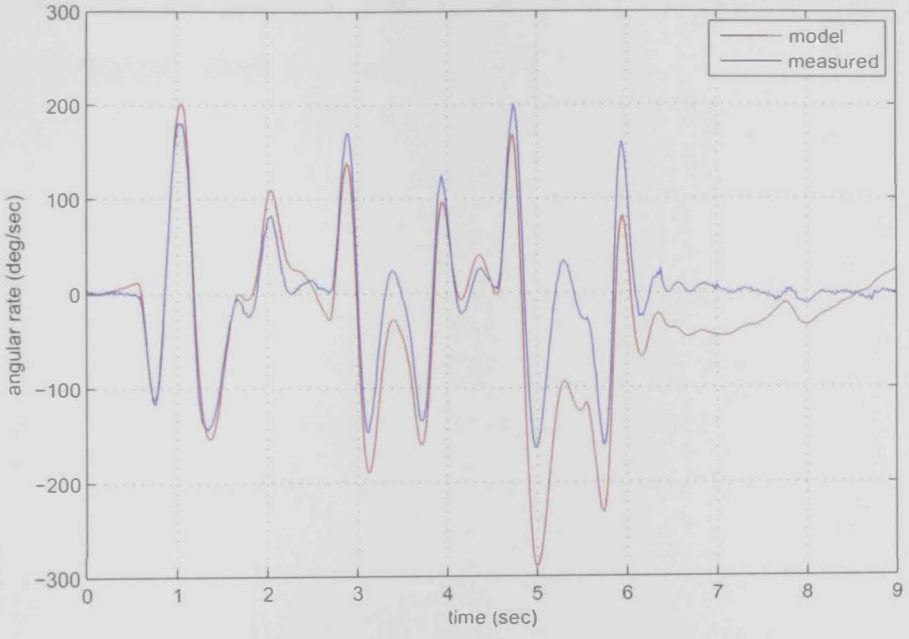


Figure 3.19: Simulink validation of the blade-thrust coefficient.

As the roll angle model of the system has an integrator, a drift is expected between the two responses over time. This drift comes from the fact that the systems Simulink model is perfect and does not have any offset in the CG and because all the motors in Simulink perform identically. However, in reality, the CG of the system is slightly shifted and the motors perform differently. This means that the actual system needs a certain trim or offset to remove the impact of these and other contributing factors. Removing this offset before injecting the input to the simulation model was not perfect, but good enough

for validation. Figure 3.19 shows that the simulated model exhibits similar behavior as the real system to a very large degree if the drift is ignored and only the magnitude of the change in signal magnitude is considered. In order to validate the experiment, the identification and parameter estimation process for blade thrust coefficient, b_t in Section 3.4 was based on the pitch axis, while the validation process was carried out mainly, but not exclusively, on the pitch axis. The reason for using different axes for validation and identification was to reduce the possibility of error in either process, as well as to verify the similarity of the models pitch and roll rates. The results were positive in both processes, with the exception of the difference in the trim between roll torque and pitch torque, which is expected.

Chapter 4

Classical Control

The purpose of this chapter is to present the defined architecture for the control system, as well as design of the classical controllers. A detailed explanation of the control architecture is given in this chapter. Then, the abstraction of the actuators for control purpose is presented, as well as the transformation between moments and thrust to motor commands. This chapter also covers the linearization of the complete system model, as well as the saturation of the system actuators. A detailed discussion of the inner-loop controllers is also given in this chapter, along with the simulation of the closed-loop system in Simulink.

4.1 Control Architecture

The control system for the quadrotor includes two main controllers, namely the inner-loop controller and the outer-loop controller, see Figure 4.1. This composition is proposed by [4] and was used in this research. The advantage of this composition is the fact that the quadrotor attitude model is independent from the translation model and that the translational model depends mainly on attitude and thrust. Therefore, such control architecture eases the overall design by splitting the complete system into two main control loops, with a dedicated controller for the independent attitude model and another for the translation model.

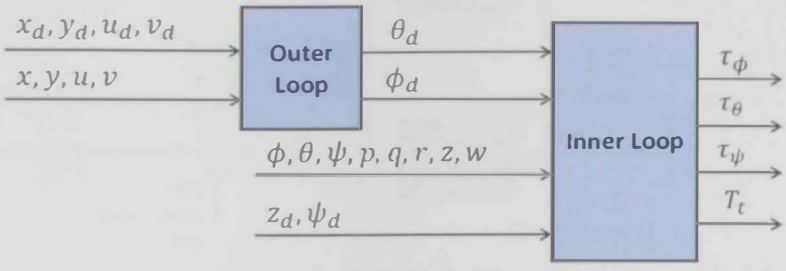


Figure 4.1: Control system architecture

The inner-loop function is there to stabilize the systems attitude and altitude, as well as to track the desired attitude and altitude commands. In turn, the outer-loop function is there to stabilize the translational motion (position and velocities) of the system, as well as to track the desired trajectory (when flying from one position to another) and to hold a desired position. The inner-loop controller interacts with the system directly by producing torques and thrust commands. The torque and thrust commands are produced according to the systems current state (attitude, body rate, climb rate, altitude), and the desired attitude and altitude. In turn, the outer-loop controller takes as inputs the current system states (current position, velocities) and desired position/velocities and produce as outputs to the inner-loop control the desired attitude (roll and pitch) to track desired position/velocities.

The closed-loop inner-loop system behaves as a body in 3D space, with a

vertical thrust to overcome the gravity. The motion of this body is produced by rotating the thrust vector on the horizontal axis and causing horizontal accelerations that build up to horizontal velocities and positions. The outer-loop is responsible for rotating the thrust vector to maintain a desired position. This indicates that the outer-loop controller is a more generic controller for systems with controllable vertical thrust (for VTOL mainly). Therefore, similarities are expected when comparing outer-loop controllers of a quadrotor with other platforms (e.g., helicopters).

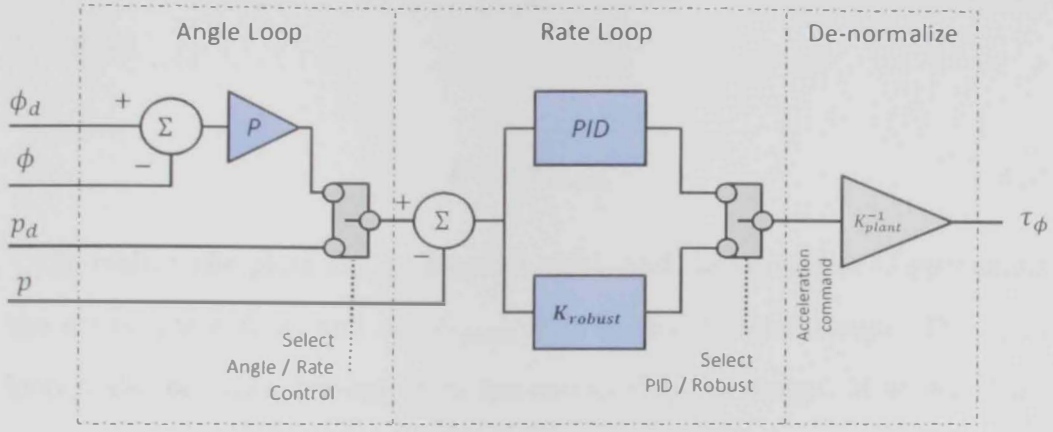


Figure 4.2: Design of the inner-loop controller

The inner-loop controller designed in this research is shown in Figure 4.2. As can be seen from the figure, the inner-loop controller includes three main parts: The angle controller (the choice of P controller will be discussed later), the angular rate controller, and denormalization. The angle controller is responsible for stabilizing the system attitude (angle) and track a reference attitude command. The angle controller produces an angular rate reference for the next controller in the design. Similarly, the angular rate controller is responsible for stabilizing the angular rate of the system and track a reference angular rate command. The angular rate controller produces normalized commands angular acceleration that are sent to the abstraction subsystem (see Section 4.2). Finally, the angular acceleration/normalized commands are denormalized to be suitable for the existing system gains. The de-normalization uses the systems inertias and mass to convert the angular acceleration com-

mands into appropriate torque commands to be fed to the system actuators.

The concept of normalization and denormalization comes from the fact that any system $G(s)$ can be decomposed into two parts, namely, the normalized plant G_{norm} with unity gain, and the plant gain K_{plant}. This decomposition is in (4.1):

$$G(s) = K_{plant}G_{norm}(s) \tag{4.1}$$

If a controller C is designed for the plant $G_{norm}(s)$, then the open loop gain L will be:

$$L = CG_{norm} \tag{4.2}$$

In reality, the plant G_{norm} does not exist, and the controller C commands the actual plant $G(s)$ and not G_{norm} that is used for the design. The open loop L also becomes different from the one used in the design. However, if the controller C is multiplied by the inverse of the plant gain (multiplied by K_{plant}^{-1}), the open loop transfer function obtained will be as in (4.3). The system transfer function in (4.3) is substituted with its decomposition to show that the open loop transfer function L becomes the same as the transfer function used in the design. This means that the controller C designed with the normalized plant G_{norm} can be used with the actual plant G. This is achieved by multiplying by the plant gain inverse. The plant gain for the application in this research is $K_{plant} = \frac{1}{I_{xx}}$.

$$\begin{aligned} L &= CK_{plant}^{-1}G(s) \\ &= CK_{plant}^{-1}K_{plant}G_{norm}(s) \\ &= CG_{norm}(s) \end{aligned} \tag{4.3}$$

The inner-loop controller is designed to be flexible for the development process. As this design was intended to be implemented and tested in experi-

mental flights, it was necessary to allow testing by focusing on each individual control loop. Hence, a switch was added to the design to allow the angular-rate reference to be taken from either one of the following two sources:

1. Angle controller:

   When the system is operated in angle control mode, the reference angle comes from the outer-loop controller, as stated earlier.

2. Externally:

   When the system is operated in angular rate mode, the external source of angular rate command, in this case, is the human test pilot operating the system.

In addition to the flexibility in selecting the source of the angular rate reference command, the angular rate controller also allows for selection of either one of the two controllers during a test. The two angular rate controllers are the PID controller and the robust controller, K_{robust}. The robust controller is designed and used only at the angular-rate loop for a couple of reasons. One of the reasons for using the robust controller in the angular rate loop only is to keep the system a simplified composition of blocks and successive control loops. The robust controller generated by algorithms is usually of high order and very non-intuitive for human adjustments, unlike the PID. Such controllers are usually treated as a black box and used without any changes or adjustments. For this reason, it makes sense to keep the black box limited in scope and not include angle control,[1] and keep the flexibility of human adjustments for the angle loop. Moreover, the angular rate loop is the most inner loop and it handles the plant dynamics directly, as well as most of the uncertainties and disturbances. If the most inner loop is designed to be very robust, with known dynamic response and relatively high bandwidth, then, closing the loops at the angle up to the position becomes much more feasible even if classical approaches are used.

[1] In some cases, developers design robust controllers to include horizontal velocity.

4.2 Abstraction

The abstraction subsystem is an interface layer between the generic control system designed for a normalized plant and the real physical platform. The function of this subsystem is to transform the torque and thrust commands produced by the controller into the platform-specific commands to obtain the desired torque and thrust - see Figure 4.3 -. The design of this subsystem is generic for quadrotors in general and is parameterized with the platform constants and parameters.

The purpose of this subsystem is to reduce the independence of the control system design from the physical system. This is achieved by combining the system-specific parameters into a single block (*abstraction*), where the control system generates torque and thrust commands. Based on the system parameters, this subsystem will produce appropriate commands for the actuators. Theoretically speaking, the control system (including gains) and abstraction subsystem can be reused as is in another quadrotor platform by only updating the abstraction subsystems with new platform parameters and modifying the inertias and mass in the de-normalization step in the inner-loop controller.

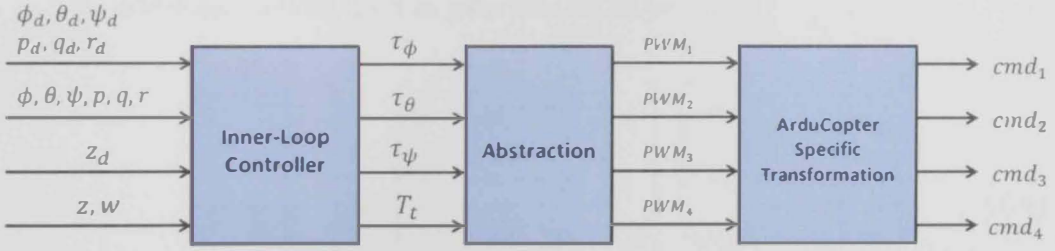


Figure 4.3: Actuator abstraction interaction with inner-loop controller and quadrotor platform.

As shown in Figure 4.4, the abstraction subsystem consists of two steps. First, the torques and thrust produced by the control system have to be coupled into motor's angular velocity. The motor angular velocity is transformed internally into a corresponding PWM command to be sent to the driver circuit of the motors. Up to this point in the signals flow, the output of this step

is generic for any quadrotor platform with a brushless speed controller that accepts PWM commands. In real life, an embedded processor or a microcontroller has to be programmed at the driver level to produce a PWM signal with a certain pulse width on its pins. As the ArduCopter environment was used in this research, this step of firmware development at microcontroller level had already been done and was only reused. Therefore, it was necessary to convert the PWM values generated so far into ArduCopter-environment commands. The block *"ArduCopter-specific transformation"* performs the conversion from the standard PWM signals into ArduCopter environment commands. Then, the generated ArduCopter-specific commands are fed to the ArduCopter environment.

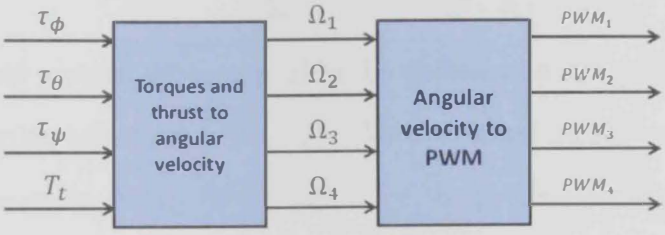


Figure 4.4: Abstraction steps to transform torque and thrust commands into PWM commands.

The actuation model used in previous sections was:

$$U = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \\ T_t \end{bmatrix} = \begin{bmatrix} 0 & -bl & 0 & bl \\ bl & 0 & -bl & 0 \\ d & -d & d & -d \\ b & b & b & b \end{bmatrix} \begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} = M\hat{\Omega} \tag{4.4}$$

This model accepts as an input the motor angular velocity. In order to generate motor angular velocity commands for certain torques and thrust, this model has to be inverted.

$$\hat{\Omega} = M^{-1}U \tag{4.5}$$

Where:

$$M = \begin{bmatrix} 0 & -bl & 0 & bl \\ bl & 0 & -bl & 0 \\ d & -d & d & -d \\ b & b & b & b \end{bmatrix} \tag{4.6}$$

The inverse of the actuation matrix, M^{-1}, was found to be - see Appendix A for detailed derivation -:

$$M^{-1} = \begin{bmatrix} 0 & \frac{1}{2bl} & \frac{1}{4d} & \frac{1}{4b} \\ -\frac{1}{2bl} & 0 & -\frac{1}{4d} & \frac{1}{4b} \\ 0 & -\frac{1}{2bl} & \frac{1}{4d} & \frac{1}{4b} \\ \frac{1}{2bl} & 0 & -\frac{1}{4d} & \frac{1}{4b} \end{bmatrix} \tag{4.7}$$

The inverted matrix, M^{-1}, is used in the abstraction subsystem to generate the motor angular velocity for the controller-generated torque and thrust commands.[1] The corresponding PWM signal for the produced angular velocity is obtained using the relations identified in Chapter 3.

Up to this point, the output of the abstraction subsystem was general for a quadrotor with PWM-driven driver circuits. Another conversion step had to be done in order to utilize the ArduCopter environment. This step involved converting the PWM command into the *cmd* command used in the environment. The transformation matrix was identified through code inspection and testing and produced the *cmd* command from a desired PWM command in order for the ArduCopter environment to produce the desired PWM signal electrically on the microelectronic pins. The benefit of including this step is to keep ArduCopter source code changes to the minimum and to utilize as much as possible from the environment.

The transformation matrix for the ArduCopter environment commands

[1] In fact, the matrix presented converts the torques and thrust into $\Omega^2_{1,2,3,4}$.

was identified as follows:

$$\begin{bmatrix} cmd_1 \\ cmd_2 \\ cmd_3 \\ cmd_4 \end{bmatrix} = \begin{bmatrix} 0 & -50 & 0 & 50 \\ 50 & 0 & -50 & 0 \\ 25 & -25 & 25 & -25 \\ 25 & 25 & 25 & 25 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 0 \\ 50 \end{bmatrix} \tag{4.8}$$

Where:

$cmd_{1,2,3,4}$ are ArduCopter environment commands for roll, pitch, yaw, and total thrust. These are interpreted in the environment as servo angles of units, $deg \times 100$, and have a range of $[-4500, 4500]$, except the thrust, which has a range of $[0, 100] \times 100$; and

$p_{1,2,3,4}$ are the standard PWM commands expressed as percentages, with the range $[\pm 50\%]$.

4.3 Linearization and Saturation

4.3.1 Attitude Linearization

From (2.13), the following assumptions could be made to achieve a linearized model:

- No aggressive maneuvers are made;
- The body is symmetrical. ($I_{xx} = I_{yy}$); and
- Only one attitude angle is excited at a time.

From the first assumption, smooth maneuvers will result in low angular rates for p, q, and r. When this is considered with the third assumption, the cross-coupling body-rotation gyroscopic effects will become very small (as it is a multiplication of these low angular rates). As a consequence, the rate of Euler angles will become almost equal to the body angular rates.

From these assumptions, (2.13) could be approximated as follows:

$$\ddot{\phi} \approx \dot{p} \approx \frac{\tau_\phi}{I_{xx}}$$

Now, if the Laplace transform is applied:

$$s^2\phi(s) - s\phi_0 - \dot{\phi}_0 = \frac{\tau_\phi}{I_{xx}}$$

Assuming the system is relaxed initially and hovering ($\phi_0 = 0$ and $\dot{\phi}_0 = 0$):

$$\phi(s) = \frac{\tau_\phi}{I_{xx}} \frac{1}{s^2}$$

The same could be applied to θ and ψ, achieving the following linear model for attitude:

$$\phi(s) = \frac{1}{I_{xx}} \frac{1}{s^2} \tau_\phi \tag{4.9}$$

$$\theta(s) = \frac{1}{I_{yy}} \frac{1}{s^2} \tau_\theta \tag{4.10}$$

$$\psi(s) = \frac{1}{I_{zz}} \frac{1}{s^2} \tau_\psi \tag{4.11}$$

4.3.2 Altitude Linearization

The general form for the system in which we decompose the linearizable dynamics from the operating points as follows:

$$Z = Z_{op} + Z_\Delta \tag{4.12}$$

With the input:

$$T_t = T_{trim} + T_\Delta$$

In order to obtain the linearized model for altitude, similarly, the Laplace transform is performed on (2.14). The result is (taking (4.12) into considera-

tion):

$$s^2 Z = \frac{T_t}{m} \cos(\phi) \cos(\theta) - g \tag{4.13}$$

In order to remove the gravity from the equation, the effect of the gravity polling the system should be compensated for automatically. This could be done by dividing the thrust into T_{trim} and T_Δ. In this case, T_{trim} will compensate for the loss of altitude due to gravity force. This yields:

$$s^2 Z = \frac{T_\Delta + T_{trim}}{m} cos(\phi) cos(\theta) - g \tag{4.14}$$

$$= \frac{T_\Delta}{m} cos(\phi) cos(\theta) + \frac{T_{trim}}{m} cos(\phi) cos(\theta) - g \tag{4.15}$$

The term containing T_{trim} is combined with gravity term to make response around the operating point Z_{op} as follows:

$$Z_{op} = \frac{T_{trim}}{m} cos(\phi) cos(\theta) - g \tag{4.16}$$

and the linearized system change about the operating point is:

$$Z_\Delta = \frac{T_\Delta}{m} cos(\phi) cos(\theta) \tag{4.17}$$

The trimmed input was computed to force the change of altitude around operating point to be zero. The computation was as follows:

$$T_{trim} = \frac{mg}{cos(\phi) cos(\theta)} \tag{4.18}$$

If hover is chosen as an operating point, the model could be obtained by simply using the previous model with ϕ and θ being zero; this yields:

$$Z = \frac{T_\Delta}{m} \frac{1}{s^2} \tag{4.19}$$

Such that:

$$T_{trim} = mg \tag{4.20}$$

4.3.3 Saturation

The saturation in the system actuation could be viewed at two levels. One level is at the motors angular rate level and the second is at the moments and thrust generated by the motors and propellers. The angular rate of the motors are within the range:

$$\Omega_n \in [0, \Omega_{max}^2], n = 1, 2, 3, 4 \tag{4.21}$$

The definition of motor saturation shown in (4.21) is absolute and makes no assumptions, unlike the saturation from moments and thrust level, in which the saturations specified are independently valid. This means that thrust saturation is valid assuming that the moments are negligible and saturation on roll moment is valid, assuming the other moments and thrust are at zero. Even though the saturation on the moments level is constrained with assumptions, it is more relevant and useful to consider saturation at the moments level, since the controller is designed to produce moments and thrust as an input to the abstract platform defined earlier.

The saturation on the thrust is:

$$T_t \in [0, 4b\Omega_{max}^2] \tag{4.22}$$

While the saturation on the moments is:

$$\tau_\phi, \tau_\theta \in [-lb\Omega_{max}^2, lb\Omega_{max}^2] \tag{4.23}$$

$$\tau_\psi \in [-2d\Omega_{max}^2, 2d\Omega_{max}^2] \tag{4.24}$$

4.4 PID Controllers Design

4.4.1 Attitude Controller

For the purpose of controlling the quadrotor, a standard PID controller was chosen as a classical control approach. This type of controller was chosen to exclude the integral part. The analysis presented in Appendix B shows that for any system with an internal integrator (quadrotor model has an internal integrator as shown in (4.25)), the controller used for this system does not necessary require an integrator action, unless input disturbance is present.The design of a PID controller is based on a normalized plant for attitude (as well as altitude, as will be discussed later). When normalizing the attitude model, the inertia as the plant gain K_{plant} is separated from the model to result in a plant with unity gain. Normalizing the attitude model by removing inertia, changes the input command from a torque command to an angular acceleration command. Similarly, when normalizing the altitude model (which accepts thrust force as an input) and separating the mass from the model, the input of the normalized model becomes translational acceleration instead of thrust force.

The control problem for attitude was solved by two methods. First, the control of the angular rate of the system as well as the vertical velocity were both solved mathematically. After closing the loop on the angular rate, another controller was added successively to the first closed-loop system for attitude control. This attitude controller was designed using the root-locus plot. It was found analytically that a PD controller would be an appropriate solution to this problem.

The normalized plant was derived by combining the motor-normalized dynamic model with the angular rate model. Applying the plant gain on the control signal, $1/I_{xx}$, and motor-propeller action transformation were handled

in a previous step, as shown in Section 4.2. The angular rate model is:

$$G_p(s) = H_{mp}(s)\dot{\phi}(s) = \frac{1}{\tau_{mp}s + 1}\frac{1}{s} \tag{4.25}$$

The naming convention that will be used from this step, onward, will be on roll attitude. However, the end result can still be applied to pitch and yaw controllers as their models dynamics are identical. The symbols $G_p(s)$ and $G_\phi(s)$ represent the angular rate plant and the roll plant, respectively. These two plants are controlled by the controllers K_p and K_ϕ , where K_p is the angular rate controller and K_ϕ is the roll controller. The plant G_ϕ should not be confused with the roll plant in this section, as the plant G_ϕ is used to model the dynamics of the angular-rate closed-loop subsystem.

The chosen controller was a standard PD controller, as stated earlier. The idea of this design is to eliminate the dynamics of the motor-propeller subsystem. This could be achieved by setting the gain of the PD controller so that the zero of the controller will be placed on the pole of the motor-propeller subsystem. Following this approach, the PD controller will be:

$$K_p = k(\tau_{mp}s + 1) \tag{4.26}$$

The loop gain in this case will be:

$$L_p = K_p G_p \tag{4.27}$$

$$= k(\tau_{mp}s + 1)\frac{1}{\tau_{mp}s + 1}\frac{1}{s} \tag{4.28}$$

$$= \frac{k}{s} \tag{4.29}$$

Therefore, the transfer function of the angular rate in the closed-loop subsystem will be:

$$CL_p = \frac{L_p}{1 + L_p} \tag{4.30}$$

$$= \frac{\frac{k}{s}}{1 + \frac{k}{s}} \tag{4.31}$$

$$= \frac{1}{\frac{1}{k}s + 1} \tag{4.32}$$

The resulting closed-loop system is a first-order system with a transient time response. The desired time constant τ_d of the closed-loop system is specified through $\tau_d = \frac{1}{k}$. The desired time constant τ_d of the closed-loop system was chosen to be $\tau_d = 0.1$ second. Therefore, the controller gain will be $k = 10$ and the controller transfer function is:

$$K_p = 10(0.1s + 1) \tag{4.33}$$

The controller K_p could be re-written in the standard PD parallel form as follows:

$$K_p = k_p + k_d s = k_p \left(\frac{k_d}{k_p} s + 1 \right) \tag{4.34}$$

Where:

$k_p = k = \frac{1}{\tau_d}$ $\qquad$ $k_d = \tau_{mp} k_p$

$k_p = 10$ $\qquad$ $k_d = 1$

Up to this stage, the system was controlled at the angular rate with the controller K_p. The next step was to continue with the cascaded control and add the controller K_ϕ to stabilize the roll angle of the system and track a commanded roll angle. The angular rate closed-loop system is represented by the transfer function in (4.30). The plant of the roll angle is obtained from (4.30) as follows:

$$G_\phi = CL_p \frac{1}{s} = \frac{1}{\frac{1}{k}s + 1} \frac{1}{s} \tag{4.35}$$

The roll-angle controller chosen was a simple P controller. This design choice was made to keep the controller simplified and to avoid unnecessary complexity in the system. The gain for the roll-angle controller was chosen using the root-locus plot. For this purpose, MATLAB was utilized to design the controller. The plot shown in Figure 4.5 is the root-locus plot for the roll-angle plant, with a proportional gain controller in closed loop. The design was guided by adding two requirements:

- A settling time of one second; and
- A damping ration of $\zeta = 0.707$

The shaded regions shown in the root-locus plot are the regions that do not satisfy the design requirements. The procedure followed was to increase the controller gain so that it is high enough to have a good dynamical response, but not too high to violate the design requirements. The controller gain was obtained using the root-locus plot as follows:

$$K_\phi = 4 \tag{4.36}$$

4.4.2 Altitude Controller

The design of the altitude controller is very similar to the attitude controller. A PD approach to control altitude is sufficient for small attitudes (according to the altitude model). However, due to the shift of plant operating point and plant input, the control system does not perform well when the system attitude is excited heavily. When the system is in a non-hover attitude ($\phi, \theta \neq 0$), the thrust vectors are divided into two parts one to gain horizontal translation and another to compensate for gravity. This is considered as a change in the input operating point (or input disturbance). As the controller has no integrator action, the closed-loop system will suffer from a steady-state error in altitude tracking. Therefore, a dynamic compensation for lost thrust is required.

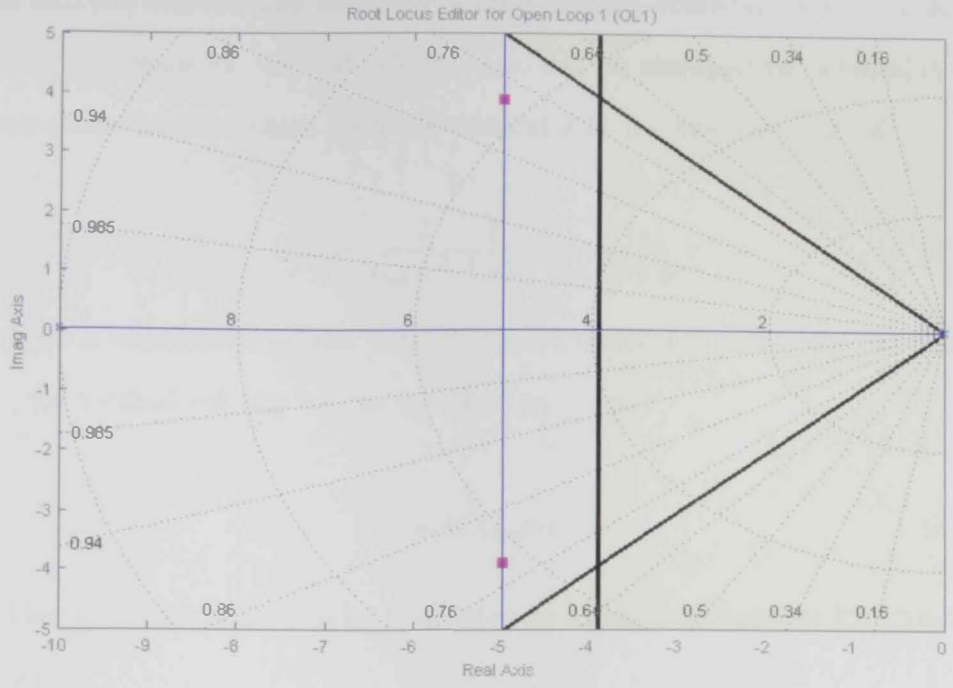


Figure 4.5: Root-locus plot for roll angle

There are two solutions for this problem. One solution could be to consider an alternative controller, in which an integrator is taken into account. This solution would have required redesigning the altitude controller as the addition of the integrator will impact the dynamic response of the closed-loop system.

An alternative solution for the inner-loop controller was opted for this research. This solution entails designing a moving operating point for the input, in which the thrust operating point is calculated instantaneously from the current attitude of the system. The general equation for the thrust operating point is known to be:

$$T_{op} = \frac{mg}{\cos(\phi)\sin(\theta)} \tag{4.37}$$

An important consideration for this approach is the impact of noisy attitude estimation. Since the attitude in this approach sets the thrust operating point, proper filtering has to take place to prevent the noise in attitude estimation from propagating into the total thrust command through T_{op}. Proceeding

with altitude control, and following similar plant normalization as in attitude control, the plant for vertical velocity can also be obtained by combining the motor-propeller subsystem with the vertical velocity integrator as follows:

$$G_{\dot{Z}} = \frac{1}{\tau_{mp}s + 1}\frac{1}{s} = \frac{1}{\tau_{mp}s^2 + s} \tag{4.38}$$

After consideration of the variable thrust operating point, the controller, $K_{\dot{Z}}$, for vertical velocity can be specified as before:

$$K_{\dot{Z}} = k(\tau_{mp}s + 1) \tag{4.39}$$

The controller, $K_{\dot{Z}}$, can be re-written in a standard parallel PID form as follows:

$$K_{\dot{Z}} = k_p\left(\frac{k_d}{k_p}s + 1\right) \tag{4.40}$$

Where:

$$k_p = k = \frac{1}{\tau_d} \qquad\qquad k_d = \tau_{mp}k_p$$

and $\tau_d = \frac{1}{k}$ is the closed-loop time constant. The closed-loop time constant was chosen to be $\tau_d = \frac{1}{2}$, $k = 2$ and the controller therefore becomes:

$$K_{\dot{Z}} = 0.2s + 2 \tag{4.41}$$

Following the successive loop closure, the closed loop up to vertical velocity has the following transfer function:

$$CL_{\dot{Z}} = \frac{G_{\dot{Z}}K_{\dot{z}}}{1 + G_{\dot{Z}}K_{\dot{z}}} = \frac{1}{s + 1} \tag{4.42}$$

The altitude controller chosen was a simple, proportional controller to ensure simplicity. Tuning the proportional gain for altitude was done using the root locus, see Figure 4.6, with MATLAB as a tool for graphical tuning. A set of two requirements were specified using MATLAB to guide the tuning

process. These requirements were:

- Damping $\xi = 0.707$; and
- A settling time of four seconds.

The proportional gain of altitude controller was found to be $k = 1$.

The vertical velocity and altitude controllers were deliberately chosen to be relatively slow compared to the attitude controllers to avoid saturating thrust. Avoiding thrust saturation is very important, since it causes all the motors to reach their maximum angular velocity, which, in turn, results in saturating commands from all the attitude controllers, even for small torque commands.

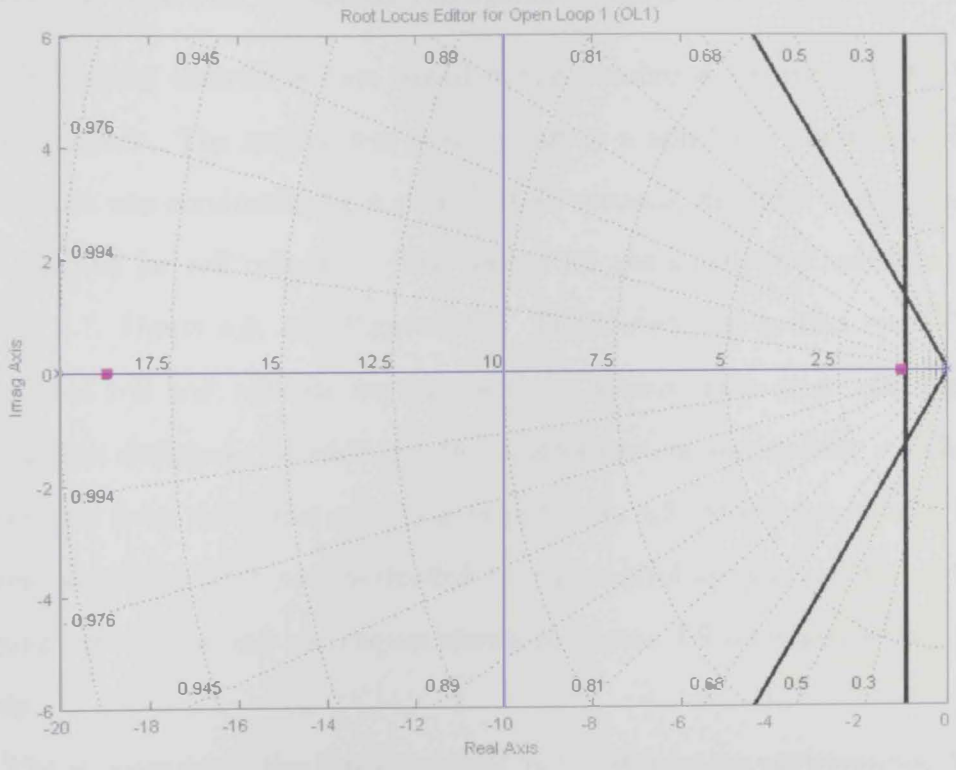


Figure 4.6: Root-locus plot for altitude

4.5 Simulation

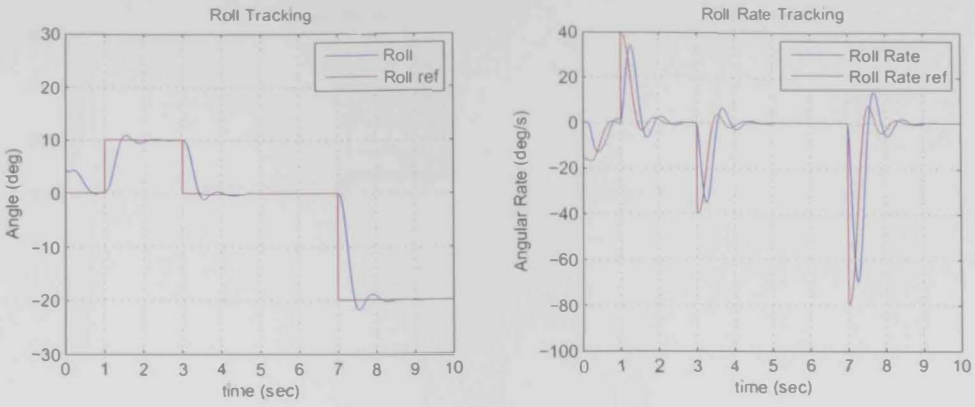
In this section, the tests for the designed inner-loop controllers are discussed. The testing was done against the complete non-linear model of the system

and was conducted by commanding different step signals on references and analyzing the system output and control inputs. The simulation was limited to only the roll angle and altitude. Simulation on pitch and yaw is expected to give very similar results to roll and, therefore, pitch and yaw were excluded from this test. The simulation started with an ideal scenario, in which all the actuators and sensors were considered to be perfect and have no noise. In the second test scenario, torque disturbances and body angular rate noise were added to the simulation environment with a frequency and magnitude which made the scenario more realistic.

4.5.1 Scenario 1: Ideal Actuators and Sensors

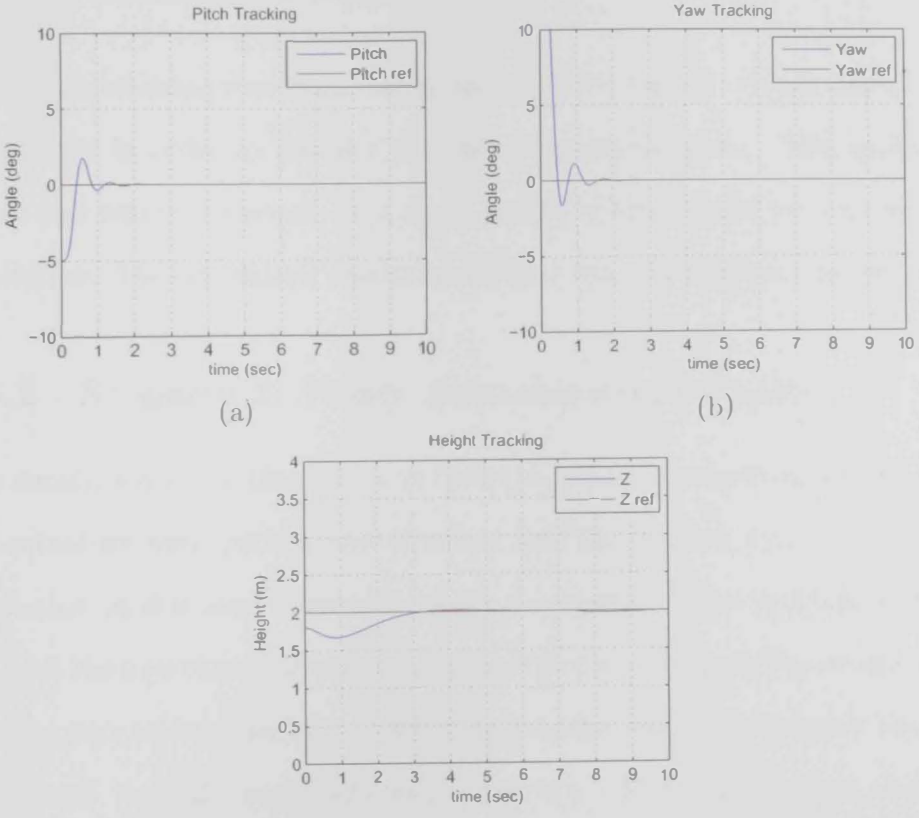
The inner-loop controllers were tested in the simulation environment for the ideal situation. The system was initially put in a non-hover condition. The simulation was conducted for a period of 10 seconds, in which a step signal was injected for roll reference. The results for the simulation are shown in Figure 4.7, Figure 4.8, and Figure 4.9b. The closed-loop system seemed to have good roll and roll-rate tracking with no steady-state error, see Figure 4.7, and as designed. In addition, the control system successfully stabilized the system from the initial condition as in Figure 4.8. Most importantly, the system actuators were not overloaded by the control system and the motor angular velocity as well as torques shown in Figure 4.9 remained within the limits.

The altitude controller was also tested in the simulation environment. The results shown in Figure 4.10 are: 4.10a for the altitude tracking plot, 4.10b for vertical velocity tracking, and 4.10c for motor angular velocity. The thrust produced by the controller did not over load the system actuators and the control system was able to stabilize the vertical velocity and altitude, and track a reference altitude successfully with zero steady-state error. Thrust compensation also seemed to work well for attitude reference steps.



(a) Roll Tracking (b) Roll Rate Tracking

Figure 4.7: Roll Tracking simulation for the ideal-case scenario

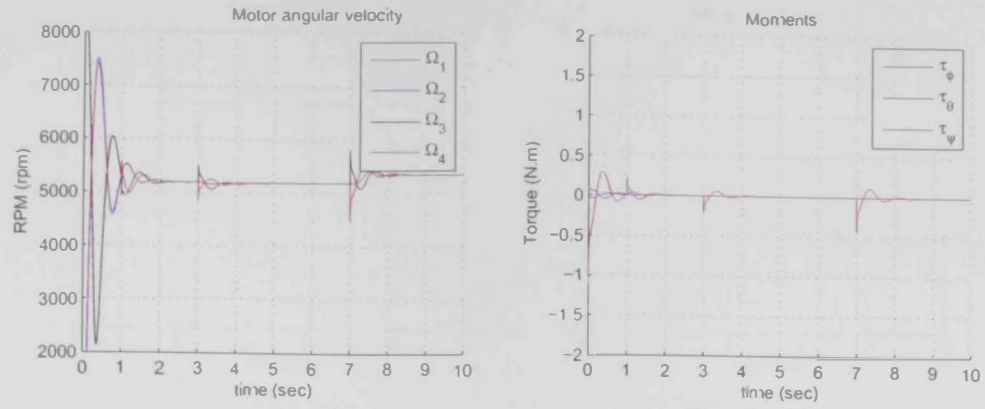


(a) (b)

(c)

Figure 4.8: Attitude and altitude stabilization

It can be noted from Figures 4.10a and 4.10b that the system was descending initially. Although the systems operating point was set properly, this behavior persisted. The reason for this short period of descent is that the motor-propeller subsystem was initialized to be at zero angular velocity.



(a) (b)

Figure 4.9: Control signals for the motors' angular velocity for attitude step references in an ideal case scenario

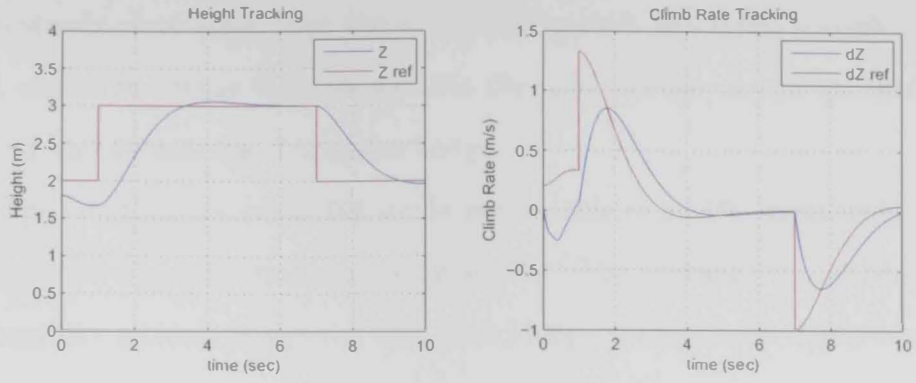
While in a hovering condition, the motors need to have a certain initial angular velocity in order to produce the thrust operating point. This problem is minor and was only present for a short period of time at the beginning of the simulation. The simulation results as a whole are still valid in general.

4.5.2 Scenario 2: Noisy Actuators and Sensors

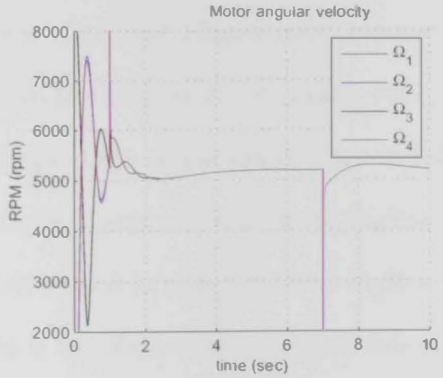
The simulation up to this step was based on an ideal simulated system, were the actuators were perfect and identical and the sensors were perfect. The simulation in this second scenario was a repetition of the previous scenario, but with the injection of actuator and sensor noise. Adding actuator and sensor noise as pure white noise was avoided. In real life, noise in general is normally distributed in both magnitude and frequency. The noise signals shown in Figure 4.11 are the noise generated by the combination of band-limited white noise and a low-pass filter (LPF) with a specified cut-off frequency. The "band-limited white noise" block generates normally distributed random numbers. The cutoff frequency of the LPF was set at 100 Hz, while the noise power was 0.0001.[1]

The results from the repeated simulation with noise injection are shown in

[1]The height of white noise power spectral density (PSD).



(a) Altitude (b) Vertical Velocity



(c) Motor angular velocity

Figure 4.10: Altitude tracking for ideal-case scenario

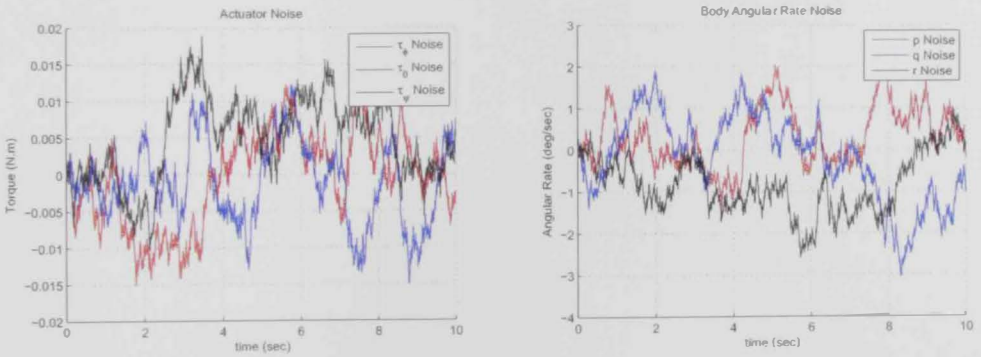


Figure 4.11: Injected input disturbance and sensor noise into the simulation environment

Figure 4.12. From these plots, it can be seen that the inner-loop controllers for attitude was unable to handle the injected noise at the input. The attitude angle plots have a more clear view of the steady-state error than the attitude-rate plots. In the first four seconds of the roll-reference command, the closed-loop system was not capable of achieving the desired reference command with

zero steady-state error. For the same time period, the input disturbance on roll, shown in Figure 4.11, shows that the noise torque on roll for that time period had an average of negative torque.

The fact that the controller would not be able to handle input disturbance was expected as explained previously in 4.4.1. The reason is that although the system has internal integrator, the closed-loop system which doesn't contain integrator will not reject actuator noise (which is an input disturbance). The simulation results show that the closed-loop system still maintains a good overall performance even in the presence of noise, except for persistent actuator noise. This is a very important consideration when taking this control system design into implementation and experimental flights. Further to simulation, the same controller was tested in experimental flights and further analysis and verification, as well as a solution to handle persistent actuator noise, will be addressed in the Implementation and Experimentation (Chapter 6).

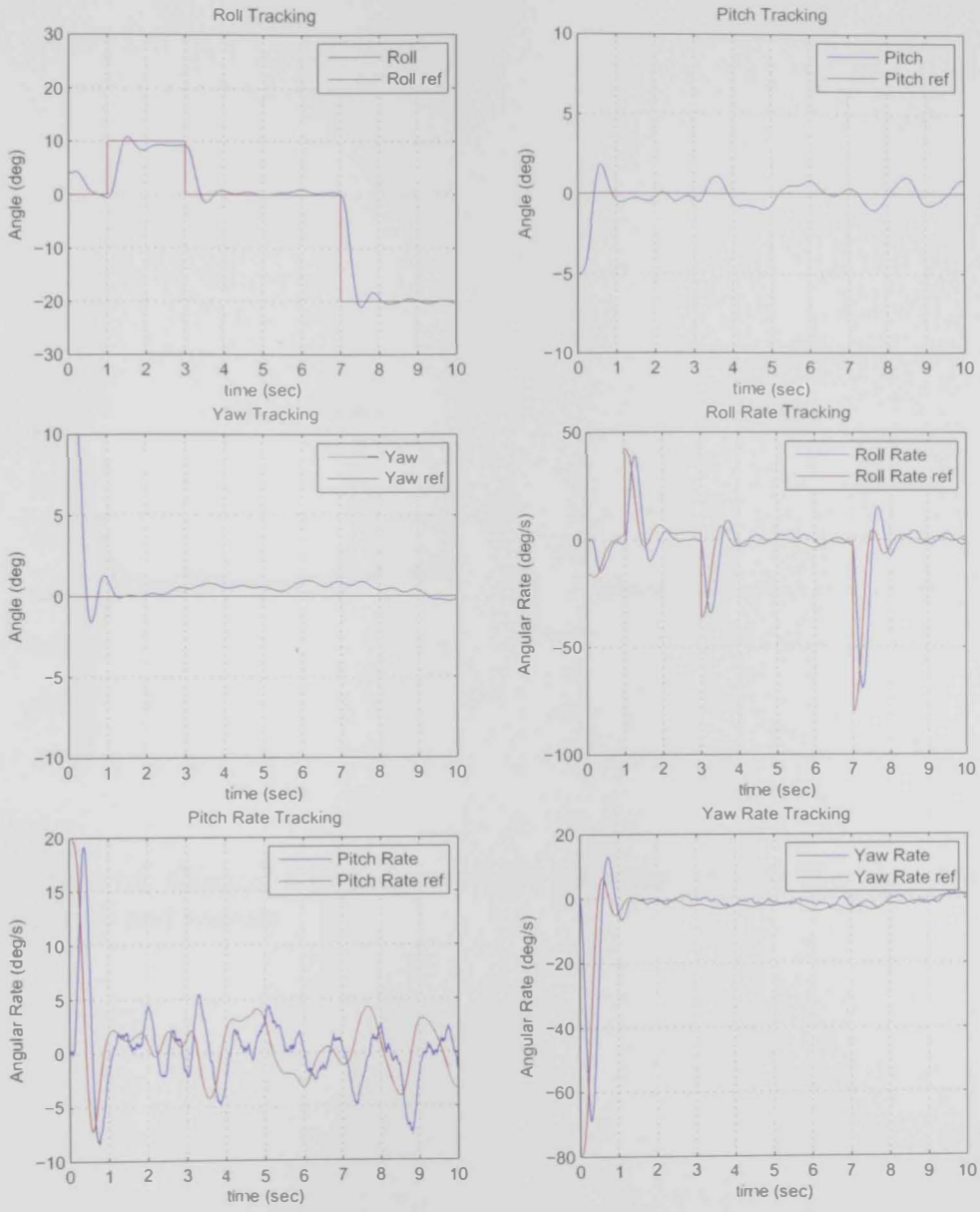


Figure 4.12: System attitude stabilization and tracking in the case of noisy actuators and sensors

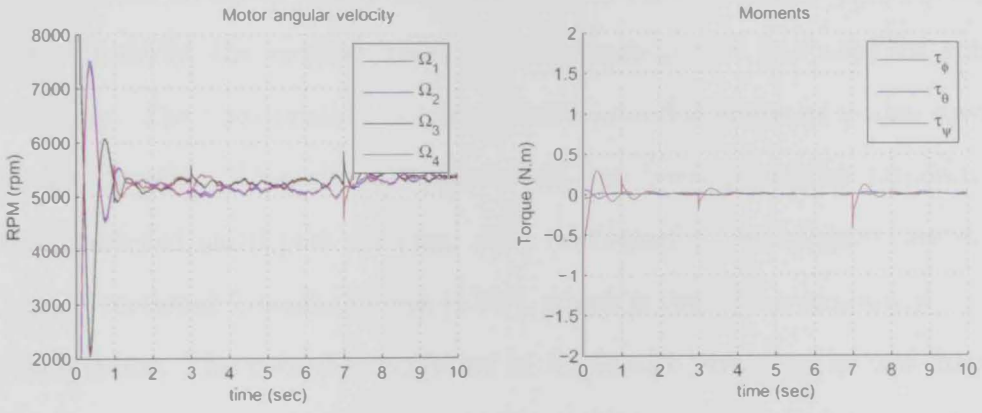


Figure 4.13: Control signals for attitude reference steps in the case of noisy actuators and sensors

Chapter 5

Robust Control

This chapter covers in details the design of the robust controller for the inner-loop. Initially, the chapter gives the justification and the need for robust controller. The classification of uncertainties and disturbances is also covered in this chapter. This chapter also covers the basic principles required for interconnected multi-port systems. As a prerequisite, this chapter covers the Linear Fractional Transformation (LFT), which is the representation of closed-loop systems. Theoretical background of H_∞ is also presented in this chapter. The chapter gives in details the problem statement of obtaining a controller using H_∞ method, and presents two known schemes to obtain the augmented plant. The $S/KS/T$ scheme used to obtain the augmented plant is discussed in this chapter, along with the design and simulation of a robust controller that is based on this scheme. Another attempt of designing a robust controller is also discussed in this chapter. The second attempt is based on GS/T scheme for obtaining the augmented plant. The theory of this scheme is discussed, and the design and simulation is presented in this chapter. The advantages and disadvantages of the two schemes is also covered. Finally, the robust controller obtained was tested in the nonlinear model. The testing covered the nominal case, and the case of noisy sensors and in presence of input disturbances. The simulation results and analysis is presented at the end of this chapter.

5.1 Introduction

In the previous chapter, a classical control technique was used to design a controller that stabilizes the system and tracks a reference signal. The design was driven by the previously identified model and the controller was tested in a simulation environment to verify the design and targeted performance. Up to this stage, the control system design procedure was based on a nominal plant and it has been assumed that the simulation model is almost identical to the real quadrotor. However, stability in nominal case is not sufficient and assuring stability is crucial in the presence of uncertainty between the real plant and the model. The goal of robust control is to design a controller that maintains system stability and performance in the presence of system uncertainty within a defined bound.

For the purpose of designing a robust controller, the H_∞ method was used in this study. This method was used to achieve a robust controller capable of stabilizing the system and tracking reference commands with good performance even in the presence of uncertainties and disturbances. Aside from the advantage of robustness that this method has over classical control techniques, this method can be used for multiple-input/multiple-output (MIMO) systems even if the system channels are coupled. These two advantages (robustness and MIMO capability) make this method a much more powerful method to use rather than the classical PID control technique.

In the process of designing a robust controller, the H_∞ method was primarily used to obtain the controller. Robust controllers are often associated with the H_∞ method, and vice versa, however, applying the H_∞ method does not necessarily ensure a robust controller. To accomplish robustness, the problem has to be formulated for robust control and the robustness specification has to be embedded into the plant. Only then the H_∞ method can be used as a mathematical tool to obtain the controller.

Uncertainties which could exist in reality are: the inaccurate model of the

system or the actuator, uncertain system parameters, degraded actuators or sensors, external disturbances (such as wind gust), changing environment (like air temperature and air density), and changing system parameters (like inertia, or mass for fuel-based air vehicles). There are two types of uncertainties, namely disturbance signals and dynamic perturbation [18]. Disturbance signals could include input and output disturbances (such as wind gusts), sensor noise, and actuator noise. Dynamic perturbation could include the discrepancy between the dynamics of the real system and the mathematical models. Perturbations of this type could come from:

- Unmodeled system dynamics (usually at high frequency);
- The impact of model order reduction; and
- Changing system parameters due to environmental changes.

A relevant example is the systems change in inertia (when adding a payload), mass, or blade thrust coefficient (which is dependent on air density).[1]

In [18], dynamic perturbations are further sub-classified into two uncertainty categories, namely unstructured uncertainty and parametric uncertainty. The former is referred to when the model uncertainties are all combined together. This category only considers the upper bound of the uncertainties maximum singular values across a known frequency. Uncertainties of this kind are usually high frequency and could result from un-modeled lags, or hysteresis. The second uncertainty category, parametric uncertainty, includes the variations of system parameters of a defined possible values range. Sources of uncertainties are inaccurate descriptions of the components characteristics, the effect of wear-and-tear, or shifting operating points.

There are couple of reasons that motivated using H_∞ to design the robust controller. Aside from the professional need, the H_∞ method can be used

[1]The air density in the laboratory environment where the identification took place is different from the air density outdoors where most of the flight tests took place.

to design a robust controller for coupled Multi-Input Multi-Output (MIMO) systems with coupled dynamics. Moreover, the method allows explicitly specifying the robustness specification, and the algorithms used produce a robust controller based on the given specifications.

5.2 Linear Fractional Transformation

For linear time-invariant (LTI) systems, the time-domain description of the system is made by the state-space representation as follows:

$$\begin{bmatrix} \dot{x} \\ y \end{bmatrix} = \begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x \\ u \end{bmatrix} \tag{5.1}$$

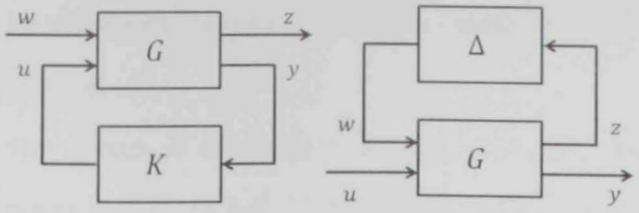
The same system can be represented in the frequency domain as follows:

$$G(s) = C(sI - A)^{-1}B + D = \left[\begin{array}{c|c} A & B \\ \hline C & D \end{array}\right] \tag{5.2}$$

The linear fractional transform (LFT) is a special case of interconnected systems where one pair of inputs/outputs of the nominal system is connected to another system. The other system connected to the nominal system could be a model of the system uncertainty, or simply the controller. Two types of LFT exist as shown in Figure 5.1. The representation of the augmented system (the combination of the two systems) would be:

$$\begin{bmatrix} \dot{x} \\ z \\ y \end{bmatrix} = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right] \begin{bmatrix} x \\ w \\ u \end{bmatrix} \tag{5.3}$$

$$G = \begin{bmatrix} G_{11} & G_{12} \\ G_{21} & G_{22} \end{bmatrix} = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right] \tag{5.4}$$



(a) Lower LFT (b) Upper LFT

Figure 5.1: Linear fractional transforms

The lower LFT is used when the plant, G, is connected to the controller, K. The upper LFT is used when the plant, G , is connected to the uncertainty block as shown in Figure 5.1b.

There are two processes applied in the design, namely analysis and synthesis. The analysis process investigates whether the uncertainty Δ can destabilizes $F_l(G, K)$ while in the synthesis process, the controller, K, is tuned to stabilize $F_u(G, \Delta)$ for all Δ satisfying the bound on the singular values.

5.3 H_∞ Overview

The notation $H_p^{m\times n}$ refers to the "[h]ardy space that contains continuous complex matrix-valued functions of the dimension $m \times n$ which have a finite H_p norm and are analytic in a certain region"[10]. According to [10], Hardy space was first introduced by the Hungarian mathematician Frigyes Riesz in 1923, who named the space after the English mathematician Godfrey Harold Hardy for his paper in 1915. The idea behind this method is to minimize the H_∞ norm of cost function. The cost function is a weighted measure of sensitivity, complementary sensitivity and a transfer matrix from disturbance to system output. The mix of weighted measures that forms the cost function is integrated over a range of frequencies.

One of the benefits of this method is its capability of handling multi-variable control in a frequency domain (unlike PID, which is limited to a SISO systems). Aside from H_∞, another method can be applied in frequency domain control,

namely H_2. The difference between these two methods is the matrix norm required to be minimized.

In general, the design of a robust controller using H_∞ has the standard configuration shown in Figure 5.2. This configuration contains the original signals for measured variables for the controller and the control signals, y and u, respectively. The signals w and z are the external input and external outputs, respectively, added by this configuration. These signals were added to be used in specifying the robustness of the system, where z is the cost to be minimized and w is used to excite the system. The objective was to minimize the H_∞ norm of T_{zw} from the input, w, to the corresponding output, z, where the H_∞ norm is defined as:

$$\|G\|_\infty = \sup_{Re(s)>0} \bar{\sigma}\left(G(s)\right) \tag{5.5}$$

Where $\bar{\sigma}$ is the maximum singular value.

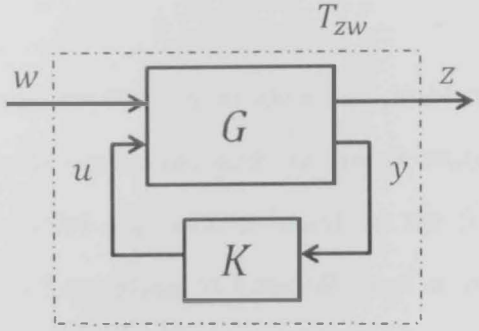


Figure 5.2: Standard H_∞ configuration

The problem statement of robust controller design using H_∞ is, for the two-port augmented plant:

$$G = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right], \quad \begin{array}{l} A \in R^{n\times n}, B_i \in R^{n\times m_i} \\ C_i \in R^{p_i\times n}, D_{ij} \in R^{p_i\times m_j} \end{array} \tag{5.6}$$

Where:

n is the order of the system.

m_i is the number of inputs for the ith port (1: w, 2: u).

p_i is the number of outputs for the ith port (1: z, 2: y).

And a controller K of order k:

$$K = \begin{bmatrix} A_K & B_K \\ C_K & D_K \end{bmatrix} \quad , \quad \begin{matrix} A_K \in R^{k \times k}, B_K \in R^{k \times p_2} \\ C_K \in R^{m_2 \times k}, D_K \in R^{m_2 \times p_2} \end{matrix} \tag{5.7}$$

was sought which would stabilize the closed-loop system:

$$T_{zw} = F_l(G, K) = G_{11} + G_{12}K(I - G_{22}K)^{-1}G_{21} \tag{5.8}$$

and which would constrain its H_∞ norm:

$$\min_K \|T_{zw}\| < \gamma_{zw} \tag{5.9}$$

Where γ_{zw} is the maximum singular value from input w to output z (transfer function T_{zw}).

There are three main approaches to solve this problem, the 1984 approach, algebraic matrix Riccati equations, and the linear matrix inequality (LMI)-based approach. The 1984 approach is based on the frequency domain and has the disadvantage of resulting in controllers of a very high order. The Riccati-based approach and the LMI-based approach are based on state-space. The disadvantage of the Riccati-based approach is that it does not allow the direct minimization of γ_{zw}, therefore, the controller is not optimal. The algorithms used nowadays are based on the Riccati equations approach and they iteratively decrease γ_{zw} until no solution exists.[1] The LMI-based approach results in an optimal solution, however, its disadvantage is that it requires higher computational power.

In order to solve the control problem, a set of assumptions had to be made.

[1]This research uses MATLAB algorithms from "Robust Control" toolbox. Both Ricatti and LMI approaches exists, but only Ricatti-based are used.

These assumptions were:

- A_1 : the matrix set (A, B_2) is stabilizable and the set (A, C_2) is detectable;
- A_2 : $\bar{\sigma}(D_{11}) < \gamma_{zw}$
- A_3 : $rank(D_{12}) = m_2$
- A_4 : $rank(D_{21}) = p_2$
- A_5 : For all ω, the two matrices:

$$\begin{bmatrix} A - j\omega I & B_2 \\ C_1 & D_{12} \end{bmatrix} and \begin{bmatrix} A - j\omega I & B_1 \\ C_2 & D_{21} \end{bmatrix} \tag{5.10}$$

  have full column and row rank, respectively; and

- A_6 : $D_{22} = 0$

The first assumption, A_1, is generic for all controllers to be able to stabilize the closed-loop system. The second assumption, A_2, suggests that the H_∞ is already solved for $s = \infty$. The assumptions A_3 through to A_5 are necessary for Riccati equations to have stabilizing solutions and assuring that the H_∞ problem is nonsingular. A_3 is more specifically meant to ensure that the input signal of the controller has an influence on the output, z. Similarly, A_4 ensures that every disturbance input, w, has an influence on the plant output, y.

Taking into consideration these assumptions, the design of the robust controller is carried out by initially specifying the robustness performance and then embedding the robustness specification with the nominal plant along with the controller, K, to form the augmented plant. The augmented plants controller would have to be tuned to minimize the cost function. Usually there are more than one aspect of the system to be considered as a cost, hence this is referred to "mixed sensitivity". The augmented plant could be constructed using different schemes. The two best-known schemes are the $S/KS/T$ scheme and the

GS/T scheme. There are few variations and extensions to these schemes. The sections 5.4 and 5.5 discuss each scheme in detail.

5.4 $S/KS/T$ Scheme

5.4.1 Theory

The $S/KS/T$ scheme, depicted in Figure 5.3, is a better known scheme for constructing the augmented plant than the scheme presented in the next section. In this scheme, the system is weighed by transfer functions in order to pass unwanted system singular values for certain frequencies and reject the singular values for another range of frequencies. Therefore, the inverse of these weighting functions shapes the system since these weighting functions define the cost to be minimized. The weighting functions in this scheme are set directly on the system transfer functions sensitivity and complementary sensitivity, hence the name $S/KS/T$. The sensitivity S and complementary sensitivity T of a system are defined as follows:

$$\begin{aligned} S &= \frac{1}{1 + KG_P} \\ T &= \frac{KG_P}{1 + KG_P} \end{aligned} \tag{5.11}$$

It can be seen from the figure that the control signal is also weighted. However, the primary weighting functions are for sensitivity and complementary sensitivity.

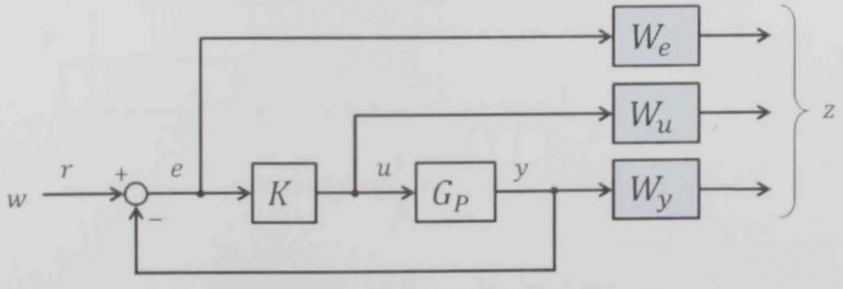


Figure 5.3: $S/KS/T$ Scheme

The weighting function, W_e, weighs the sensitivity, S, of the system. In general, the singular value of the sensitivity function is required to be small at low frequencies and relatively large at high frequencies. Therefore, as a guideline, the singular value of the weighting function, W_e, is set to be high at a low frequency (for disturbance rejection) and small at a high frequency. The bandwidth of the resulting closed-loop system is driven by the cutoff frequency of W_e. The weighting function, W_y, weights the complementary sensitivity and, therefore, W_y is usually low at low frequencies and high at high frequencies symmetric to W_e. The weight, W_u, which stands for the control signal, fulfills A_3, which assumes the feed-through in the augmentation. The weighting in the control signal is not usually dominant and is set to be a small constant which will not influence the norm. It is important to note that all the weighting functions have to be stable, otherwise, A_1 will be violated and the system will be unstable and uncontrollable.

The plant shown in Figure 5.3 was augmented and rearranged slightly, without changing the connections between the functions, to conform to the standard H_∞ configuration presented earlier in 5.2. The augmented plant in Figure 5.4 is for the $S/KS/T$ scheme using the standard configuration of H_∞.

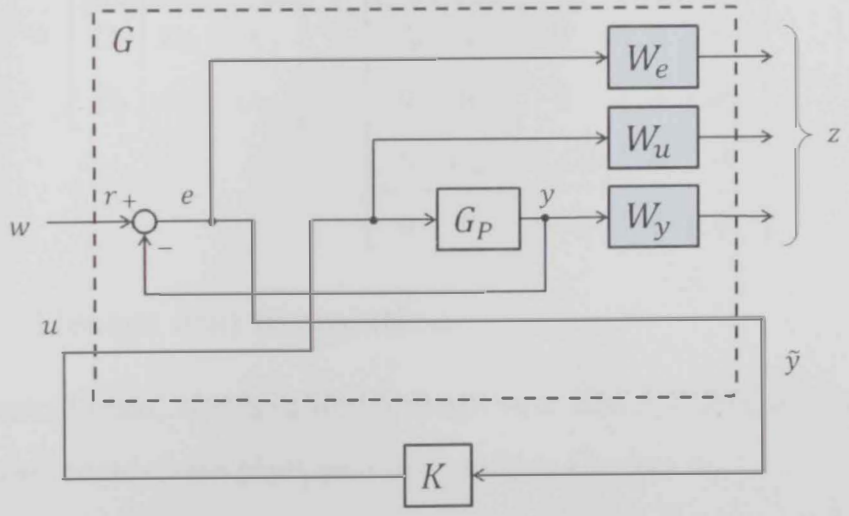


Figure 5.4: Augmented plant using the $S/KS/T$ scheme

The cost function for this scheme is:

$$T_{zw} = \begin{bmatrix} W_e S_e \\ W_u K S_e \\ W_y T_e \end{bmatrix} \tag{5.12}$$

Where the augmented system G can be represented in the frequency domain by (obtained from [10]):

$$G = \left[\begin{array}{c|c} W_e & -W_e G_P \\ 0 & W_u \\ 0 & W_y G_P \\ \hline I & -G_P \end{array}\right] \tag{5.13}$$

The notation G_P is used for the nominal plant. The state-space representation of the augmented plant, G, in the time domain is:

$$G = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right] = \left[\begin{array}{ccc|cc} A_e & 0 & -B_e C_P & B_e & 0 \\ 0 & A_y & B_y C_P & 0 & 0 \\ 0 & 0 & A_P & 0 & B_P \\ \hline C_e & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & D_u \\ 0 & C_y & D_y C_P & 0 & 0 \\ \hline 0 & 0 & -C_P & I & 0 \end{array}\right] \tag{5.14}$$

5.4.2 Design and Simulation

The nominal plant, G_P, used in the design with $S/KS/T$ scheme is also the normalized angular rate plant used previously in Chpater 4:

$$G_P = \frac{1}{s(\tau_{mp} s + 1)} = \frac{1}{0.1 s^2 + s} \tag{5.15}$$

The weighting functions, $W_e = W_s$ and $W_y = W_t$, were set using the tem-

plates (5.16) and (5.17), respectively. The value of M determined the sensitivity peak, where the value of A determined the maximum allowed steady-state error when shaping the sensitivity. A typical value of M is $6dB$. The desired bandwidth of the system was determined by setting ω_0 to a desired value.

$$W_s = \frac{s/M + \omega_0}{s + \omega_0 A} \tag{5.16}$$

$$W_t = \frac{s + \omega_0/M}{As + \omega_0} \tag{5.17}$$

The scheme shown so far is for shaping the sensitivity at the output of the system. Another variation also exists for shaping the sensitivity at the input of the system, as well as an extension to this scheme [10].

Setting and tuning the weighting functions took many iterations and the values achieved in [13] were used as a starting point.

The following parameters for the template of sensitivity weighting function were chosen:

- $\omega_0 = 4rad/s$
- $A = -40dB$
- $M = 6dB$

Therefore, the weighting function, W_e, was:

$$W_e = \frac{0.5012s + 4}{s + 0.04} \tag{5.18}$$

Similarly, the following parameters for the template of complementary sensitivity were chosen:

- $\omega_0 = 10rad/s$
- $A = -40dB$

- $M = 6dB$

And the resulting weighting function, W_y, became:

$$W_y = \frac{s + 5.012}{0.01s + 10} \tag{5.19}$$

The weighting on the control signal was set to zero, $W_u = 0$, so that it could not influence the norm. The singular plot shown in Figure 5.8b is for the weighting functions W_e and W_y.

The augmented plant was constructed in MATLAB using the earlier defined weighting functions and nominal plant, see Appendix C. Running the script resulted in the controller, . The algorithm minimized the value of γ_{zw} to be $\gamma_{zw} = 0.73639$, and it took a total of 80 iterations. The resulting closed-loop system sensitivity and complementary sensitivity functions were plotted in the same singular value with the weighting functions, as can be seen in Figure 5.8a. The algorithms successfully generated the required controller and the weighting functions shaped the closed-loop system sensitivity and complementary sensitivity as can be seen from the singular value plots in Figure 5.8.

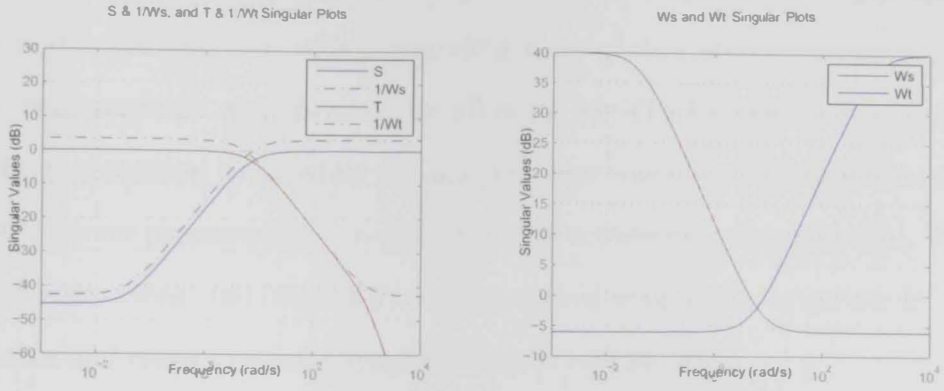
The resultant controller is:

$$A_K = \begin{bmatrix} -0.03963 & 0 & 0 & 0 \\ 0 & -1000 & 0 & 10 \\ 1.878 \times 10^6 & -4.122 \times 10^7 & -3289 & -2.159 \times 10^6 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad B_K = \begin{bmatrix} 10.02 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$
$$C_K = \begin{bmatrix} 1.873 \times 10^5 & -4.112 \times 10^6 & -327.1 & -2.153 \times 10^5 \end{bmatrix} \quad D_K = 0 \tag{5.20}$$

The controller dynamics were analyzed with the support of MATLAB. The

controller poles and zeros were found to be:

$$poles = \begin{bmatrix} -0.034 \\ -2546 \\ -871.6 \\ -871.6 \end{bmatrix}, \quad zeros = \begin{bmatrix} -1000 \\ 0 \\ -10 \end{bmatrix} \tag{5.21}$$



(a) S, T, W_e^{-1}, W_y^{-1} (b) W_e and W_y

Figure 5.5: Singular value plots

The controller generated with the augmented plant on this scheme had two major problems. The first problem was plant inversion. The controllers generated with this scheme have a tendency to invert the nominal plant, as has been proven by [10]. From the zeros and poles of the controller obtained by MATLAB in (5.21), it can be seen that the original system pole $(\tau_{mp}s + 1) = 0.1s + 1$ (pole at -10) is in fact a zero in the controller, K. The inversion of the plant in the control system is discouraged, especially when the system has zeros or poles near the imaginary axis. The reason for this is that the uncertainties in the system could cause the location of poles and zeros to shift.

Another problem with the generated controller relates to the feasibility of implementing it. The controller, K, is a dynamical system on its own and it has states that propagate according to its dynamics (poles) and inputs (error signals). The dynamics of the controller were found to be very fast by investigating the poles of the controller.

In the experimental part of this research, the controller, K, was implemented on a microcontroller and its states were propagated in discrete time with a sampling time of 0.01 seconds (100 Hz). The controllers poles presented in (5.21) required a much higher sampling rate than 100 Hz, and the controller as a dynamical system became very unstable when implemented and propagated in a 100 Hz embedded software loop. The time response shown in Figure 5.6 is for the step response of the closed-loop system for a continuous case and a discrete case with a sampling time of 0.01 seconds. The continuous time-response was obtained by allowing MATLABs functions to decide on the propagation time, while the discrete case was obtained using forward-Euler discrete propagation (integration) with a constant sampling time. The closed-loop system and controller, K, propagated with a 100 Hz system is very unstable and cannot be implemented into the real system.

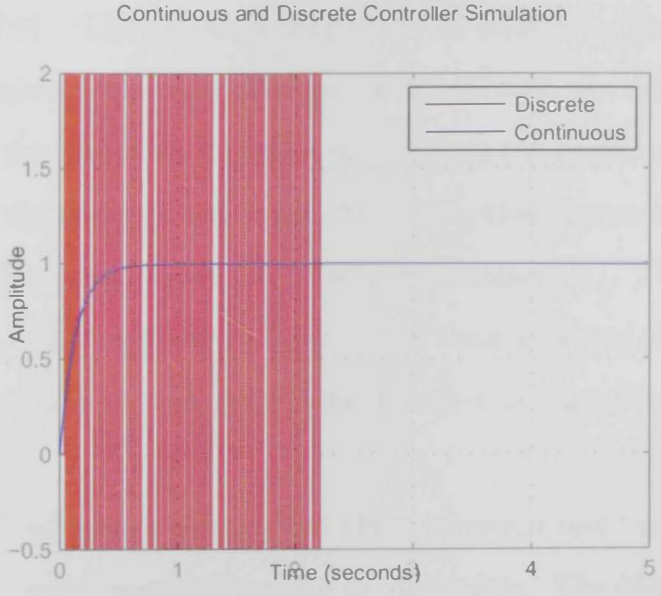


Figure 5.6: Continuous vs. discrete simulation of a closed-loop system

The problem of inverting the plant is a generic and known problem of this scheme. The problem with the fast dynamics of the controller could be related to how the weighting functions are set and it may be possible to slow down the controller dynamics by tuning the weighting functions and re-running

the algorithms to obtain a better controller. Another way to solve these two problems (inverting the plant and fast dynamics) is using the GS/T scheme, as explained in the next section.

5.5 GS/T Scheme

5.5.1 Theory

The GS/T scheme is a less commonly used scheme than the $S/KS/T$ scheme. This scheme is used in [35] to stabilize and control a helicopter. The scheme used in [35] is an extended variation GS/T scheme and is based on a MIMO-coupled linearized system. This scheme has an advantage over the S/KS/T scheme in that it overcomes the problem of plant inversion.

Another solution to prevent plant inversion is proposed by Kwakernaak as stated in [10]. The proposed solution is to include in the poles of the plant in the sensitivity weight. However, this approach has three main drawbacks. First, this approach increases the order of the augmented plant and, consequently, the resulting controller. Moreover, this approach could violate a previous assumption in case the system is unstable (A_1). The violation of assumption A_1 happnes as the unstable part of the augmented plant is not controllable. In addition to these drawbacks, Kwakernaaks approach is limited to SISO systems.

This GS/T scheme overcomes the plant inversion problem by weighting sensitivity and complementary sensitivity differently. The idea is to include the plant in the sensitivity weight. The GS/T scheme is presented in Figure 5.7, where it can be seen that the scheme is similar to the $S/KS/T$ scheme with shaping at the input,[1] where the sensitivity is weighted at y to include the plant. The scheme has an extra input, v, to allow exciting u.

The weighting functions, W_n, W_y, and W_d, are used to set the augmented

[1]The $S/KS/T$ scheme presented in this research shapes at the output. Refer to [10] for $S/KS/T$ with shaping at the input.

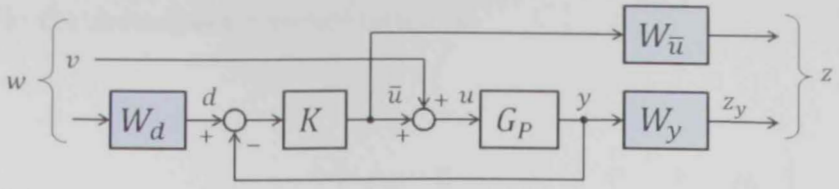


Figure 5.7: GS/T scheme

plant. The weighting function $W_{\bar{u}}$ shapes the sensitivity, while W_y shapes the complementary sensitivity. The weight, W_d, is used to fulfill the assumption A_4 ($rank(D_{21}) = p_2$). Similar to how the scheme was rearranged to derive the augmented plant, the same process can be applied to this scheme to obtain the augmented plant. The cost function in this case becomes (see [10]):

$$T_{zw} = \begin{bmatrix} -W_{\bar{u}}T_u & W_{\bar{u}}S_uW_d \\ W_yG_PS_u & W_yT_eW_d \end{bmatrix} \tag{5.22}$$

It can be seen from (5.22) that the plant is included in the cost and therefore, no plant inversion is expected in the controller. The cost function is of rank two, since there are two w inputs and two z outputs.

The augmented plant G for this scheme is represented as follows in the frequency domain:

$$G = \left[\begin{array}{cc|c} 0 & 0 & W_{\bar{u}} \\ W_yG_P & 0 & W_yG_P \\ \hline -G_P & W_d & -G_P \end{array}\right] \tag{5.23}$$

While the state-space representation is:

$$G = \left[\begin{array}{c|c:c} A & B_1 & B_2 \\ \hline C_1 & D_11 & D_12 \\ \hdashline C_2 & D_21 & D_22 \end{array}\right] = \left[\begin{array}{ccc|cc:c} A_{\bar{u}} & 0 & 0 & 0 & 0 & B_{\bar{u}} \\ 0 & A_y & B_yC_P & 0 & 0 & 0 \\ 0 & 0 & A_P & B_P & 0 & B_P \\ \hline C_{\bar{u}} & 0 & 0 & 0 & 0 & D_{\bar{u}} \\ 0 & C_y & D_yC_P & 0 & 0 & 0 \\ \hdashline 0 & 0 & -C_P & 0 & D_d & 0 \end{array}\right] \tag{5.24}$$

5.5.2 Design and Simulation

The weighting functions can be set using the same templates defined previously, (5.16) and (5.17). The sensitivity weighting is set with a cutoff frequency of $\omega_0 = 4\pi rad/s$, which corresponds to the desired bandwidth for the closed-loop system. This value is set to have a response time of 0.5 seconds. The maximum allowed steady-state error was chosen to be 0.01, which is typical, and therefore, $A = -40dB$. The sensitivity peak, M, is typically $M = 6dB$, but the chosen value was slightly higher than the typical value ($M = 10dB$). Similarly, the weighting on complementary sensitivity was set with the same DC gain and a high frequency gain ($M = 10dB, A = -40dB$), but with a cutoff frequency of $\omega_0 = 16\pi rad/s$. This weighting function setting was achieved after many iterations. At this stage, the controller (without considering W_d) was saturating the actuators. After fixing the weighting functions on sensitivity and complementary sensitivity, the cost on W_d was added gradually to reduce the actuator saturation. The weighting functions were finally set as follows:

$$W_{\bar{u}} = W_s = \frac{0.3162s + 12.57}{s + 0.1257} \tag{5.25}$$

$$W_y = W_t = \frac{s + 15.9}{0.01s + 50.27} \tag{5.26}$$

$$W_d = 0.005 \tag{5.27}$$

The singular value plot of the weighting functions is shown in Figure 5.8b. The augmented plant was constructed in MATLAB using the presented weighting functions and with the definition presented in (5.24). The script used to construct an augmented plant using GS/T scheme is written in Appendix D, and used in the script in Appendix C. The same functions were used in MATLAB to find the controller that solves the H_∞ problem for the augmented plant presented in this section. The algorithm took 45 iterations and minimized the value of γ_{zw} to 0.5. The resultant controller is:

$$A_K = \begin{bmatrix} -50.04 & -0.4845 & -0.1355 & -1.816 \\ 0 & -0.006449 & 0 & 0.0003236 \\ 4977 & -0.4268 & -10.14 & -1461 \\ 0 & 0.003704 & 1 & -54.03 \end{bmatrix} \quad B_K = \begin{bmatrix} 0 \\ -706.4 \\ -1.028 \times 10^5 \\ -3806 \end{bmatrix}$$

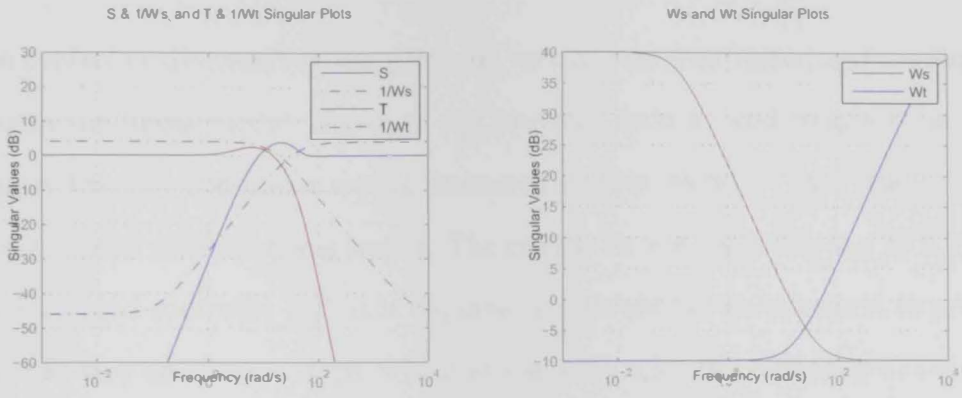
$$C_K = \begin{bmatrix} 7.065 & -0.0006879 & -0.0001923 & -0.002578 \end{bmatrix} \quad D_K = 0 \tag{5.28}$$

Which has the following poles and zeros:

$$poles = \begin{bmatrix} -55.3772 \\ -29.4139 + 42.0857i \\ -29.4139 - 42.0857i \\ -0.0065 \end{bmatrix}, \quad zeros = \begin{bmatrix} -5026.55 \\ -9.817 \\ -3.3 \end{bmatrix} \tag{5.29}$$

It can be seen that the resultant controller does not invert the plant and that it can propagate in the 100 Hz embedded system. The singular value plots in Figure 5.8 shows that the closed-loop system sensitivity and complementary

sensitivity were successfully shaped by the weighting functions. The closed-loop system was simulated in continuous time and in discrete time with a fixed time step of $0.01s$. The step responses of both continuous and discrete time are presented in Figure 5.9. The plot shows that the closed-loop system with the generated controller is stable and can be taken into experimental flights.



(a) S, T, $W_{\bar{u}}^{-1}$, W_y^{-1} (b) $W_{\bar{u}}$ and W_y

Figure 5.8: Singular value plots

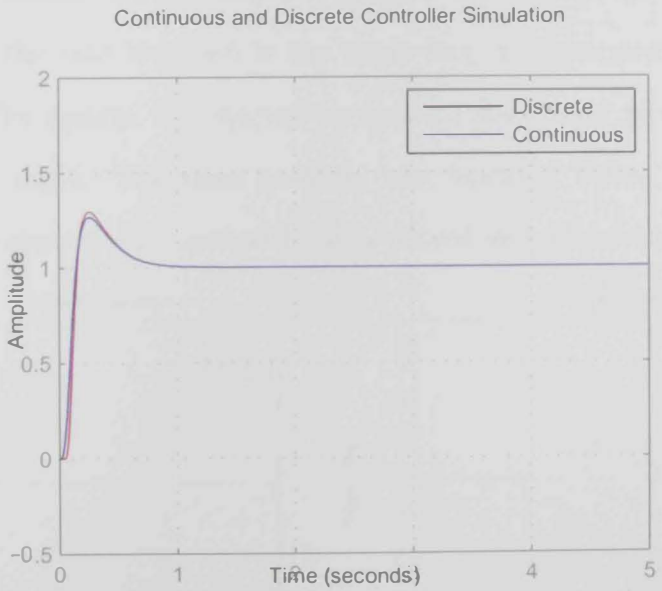


Figure 5.9: Continuous vs. discrete simulation of closed-loop system

Controllers generated from the scheme are also not perfect and do have a drawback as well. Generally, the standard GS/T scheme results in controllers

that have worse tracking abilities than the $S/KS/T$ scheme. Better tracking can be achieved by the extended variation of this scheme and the design of 2DoF controllers, as shown in [35].

5.6 Nonlinear Simulation

The controller discussed in the previous section was first tested and analyzed against the linear model, shown in the previous section, and then was tested against the full non-linear model discussed in this section. Only the GS/T scheme-based controller was tested. The reason for not including the $S/KS/T$ scheme-based controller is that it requires the simulation environment to propagate at very small time steps, which amounts to a lot of processing to obtain results for only a short period. Furthermore, the controller for this scheme cannot be implemented and is not considered as robust as the GS/T scheme.

The GS/T scheme-based controller was tested for the same scenario as the PID in the Classical Control chapter (Chapter 4). The robust controller was used only for the rate loop, while the angle loop was designed to have a P controller.[1] The system was started in nonzero state, and given a reference signal for roll angle. The same scenario was repeated twice, once for ideal actuators and sensors and once with noisy actuators and sensors.

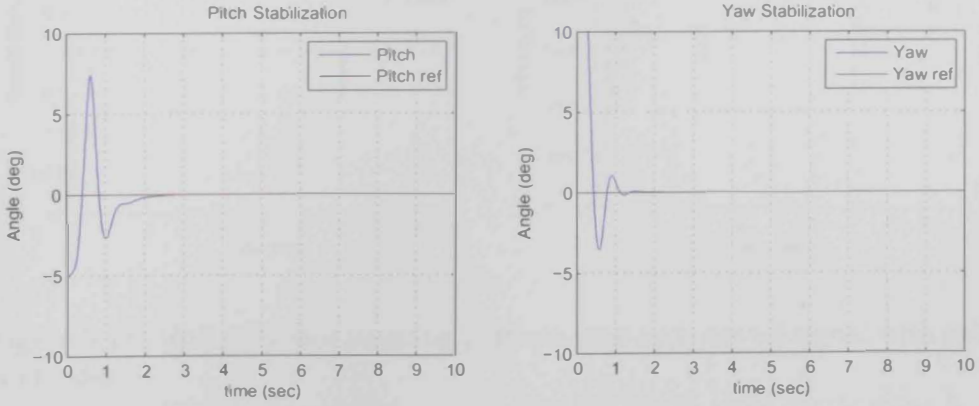


Figure 5.10: Attitude stabilization with robust controller.

[1] A detailed discussion of this is in Section 4.1 in Chapter 4.

As can be seen from the pitch and yaw plots in Figure 5.10, the controller successfully stabilized the system and kept the system in zero state (hover). It can also be noticed during the first second that the pitch stabilization was not properly handled by the controller. The reason for this was the actuator saturation as shown in Figure 5.11. During the first second -see Figure 5.12-, the error from the various state variables accumulated to a relatively large command, which caused the actuators to saturate.[1] This is not to be considered an issue since the saturation is expected when the entire system is excited. What is important to note here, is that the controller does not saturate the system when individual variables are handled, such as during roll tracking, as shown in Figure 5.11.

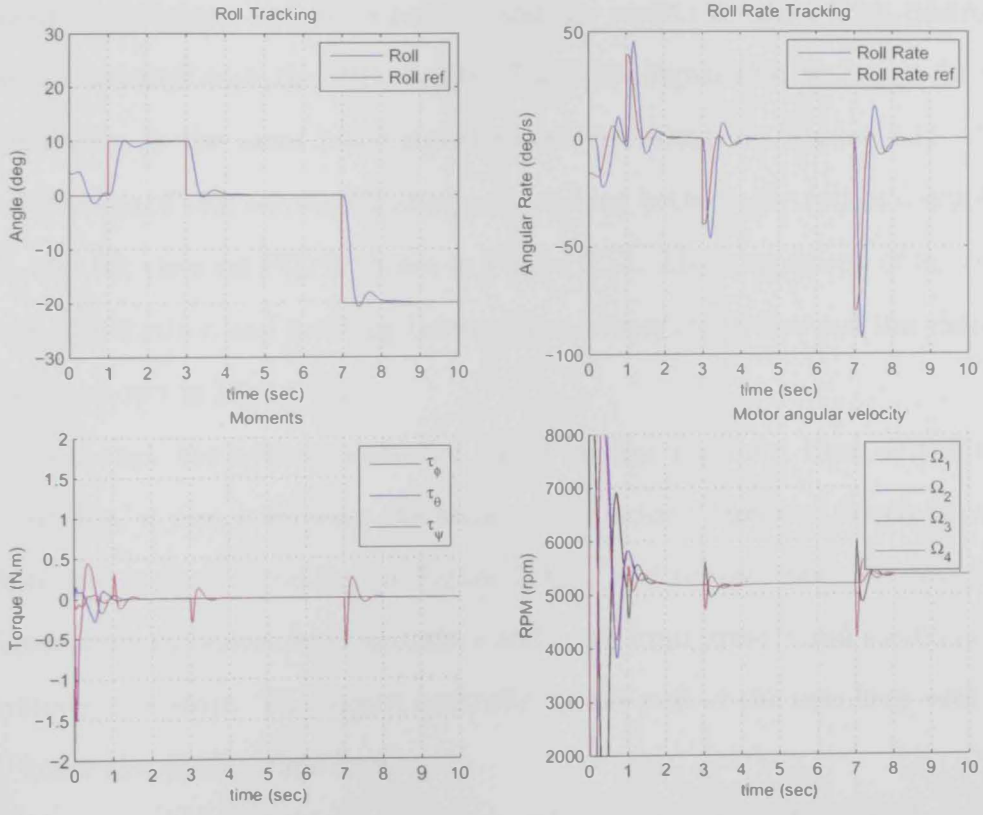


Figure 5.11: Roll reference tracking performance and control signal with robust controller.

The same scenario of reference tracking and stabilization was repeated for

[1]The data shown in Figure 5.12 is for the motors command before applying the saturation. The saturation value is shown in dashed line.

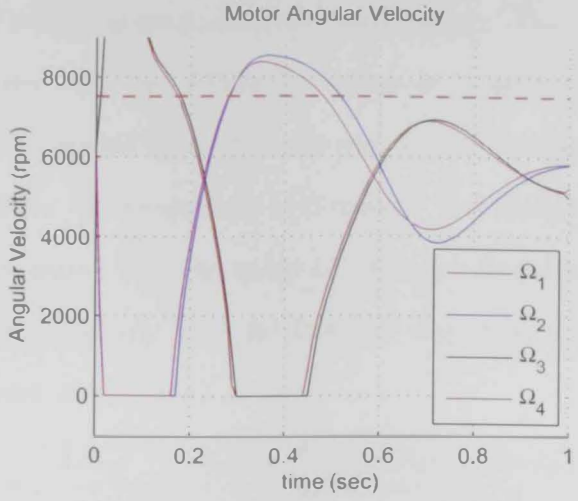


Figure 5.12: Plot of motors short saturation during the first second of the simulation.

input disturbances and noisy sensors and the results for the robust controller were compared with the PID results. The noise signal that was used for this simulation is the same noise signal shown previously in Figure 4.11. The comparison of attitude stabilization and tracking between the robust controller, K, and the classical PID is shown in Figure 5.13. The comparison of attitude-rate stabilization and tracking between the robust controller and the classical PID is shown in Figure 5.14.

Although the robust controller has a similar response time as the PID controller, it also overcomes the steady-state error. This can clearly be seen from the attitude tracking in Figure 5.13. The robust controller responds faster to disturbances from actuators and noise from sensors and maintain the systems zero state. The robust controller works well at the rate loop with the P controller in the angle loop.

A better view of the performance difference between the two controllers can be seen in Figure 5.15. The histogram of the error signal is obtained, and the results for both controllers is shown in the same plot in Figure 5.15. The data used in this figure is obtained from another simulation run in which the two controllers were stabilizing the nonlinear system for 100 seconds, and in

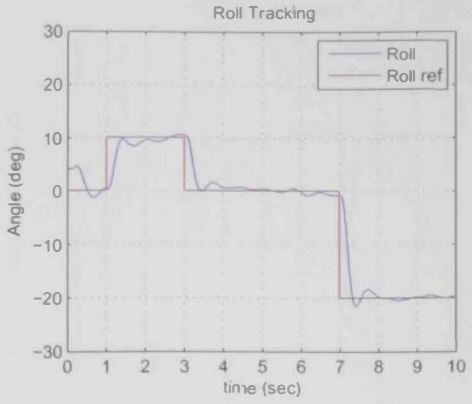
the presence of sensor noise and input disturbances. The results presented in Figure 5.15 shows that the robust controller is capable of keeping the error in a much smaller bound than the PID controller. Furthermore, the norm of the error signal for the same data is obtained and shown in Table 5.1. The data in the table shows that the norm of the error signal in case of the robust controller is approximately half the norm of the error signal in case of PID controller for most of the loops.

Loop	Norm (PID)	Norm (Robust)
Roll rate	315.27	178.60
Pitch rate	332.82	183.61
Yaw rate	126.12	84.54
Roll	68.02	33.56
Pitch	73.91	35.03
Yaw	30.72	21.76

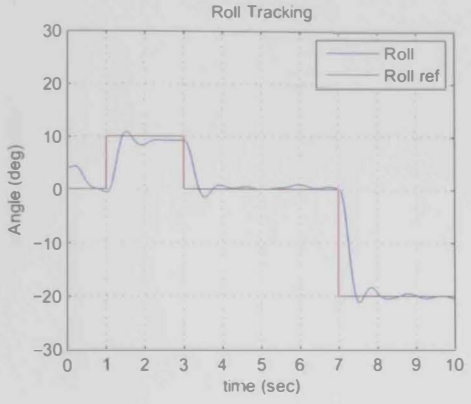
Table 5.1: Error signal norm comparison for PID controller and robust controller.

The results of the tests in the simulation environment with the non-linear model are not sufficient for comparing the two kinds of controllers. One of the reasons for this is that the non-linear model was not identified completely.[1] Furthermore, the real platform is the complete system which includes all the dynamics and disturbances impacting it. Testing with the real platform, as will be shown in the Implementation and Experimentation (Chapter 6), will indicate the controllers performance in the presence of the dynamics that were neglected in the design, the model discrepancy, and changing parameters.

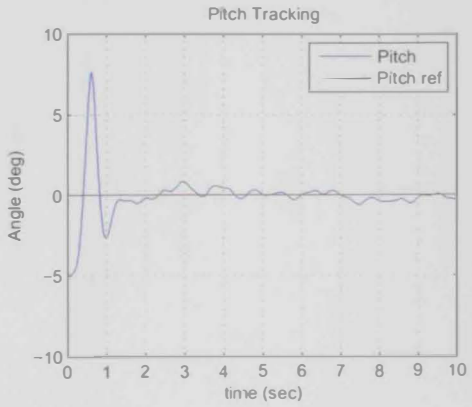
[1] The gyroscopic effect coefficient of the blades is not included in the identification.



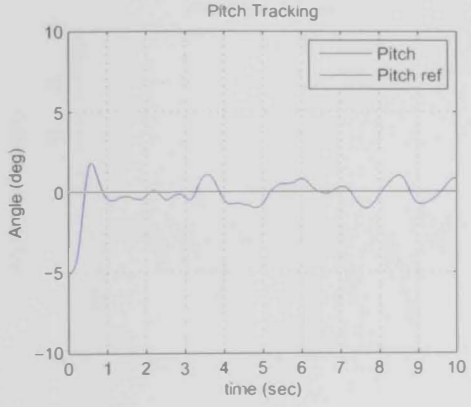
(a) Roll tracking for K_{robust}



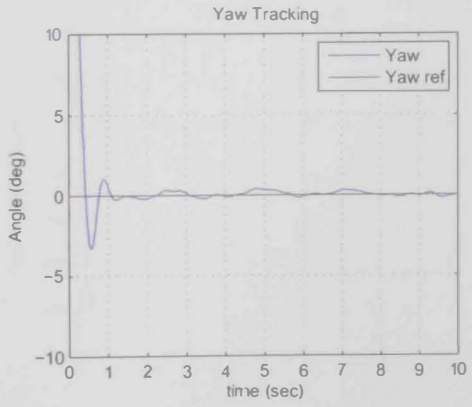
(b) Roll tracking for PID



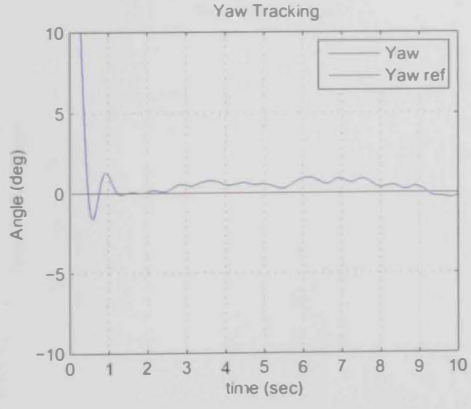
(c) Pitch tracking for K_{robust}



(d) Pitch tracking for PID

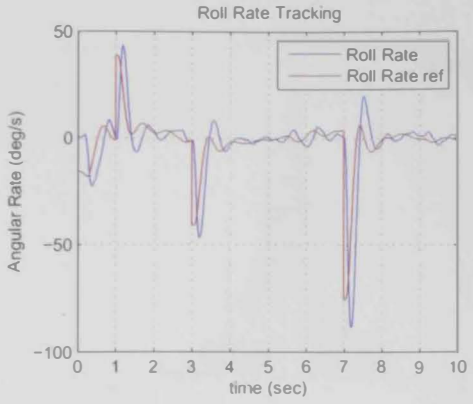


(e) Yaw tracking for K_{robust}

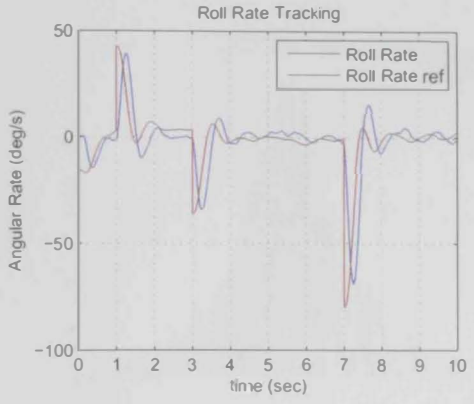


(f) Yaw tracking for PID

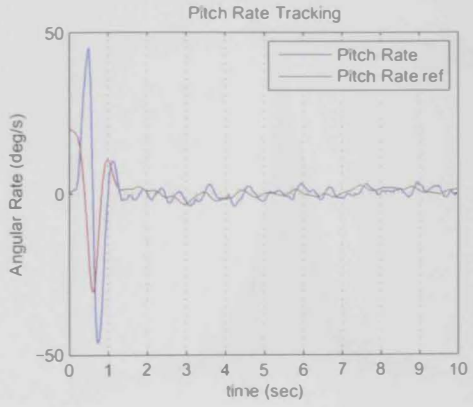
Figure 5.13: Robust controller vs. PID controller - attitude tracking comparison



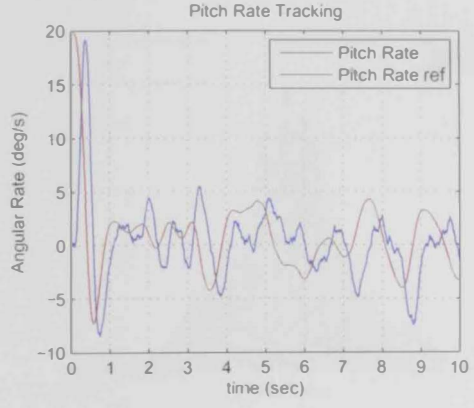
(a) Roll rate tracking for K_{robust}



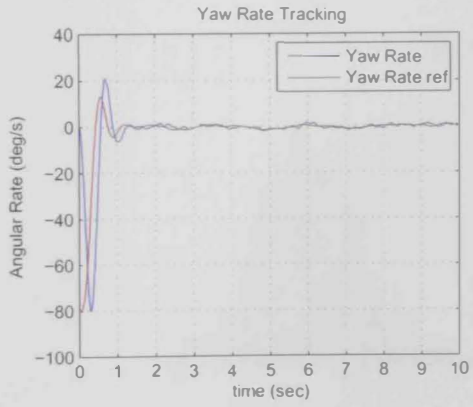
(b) Roll rate tracking for PID



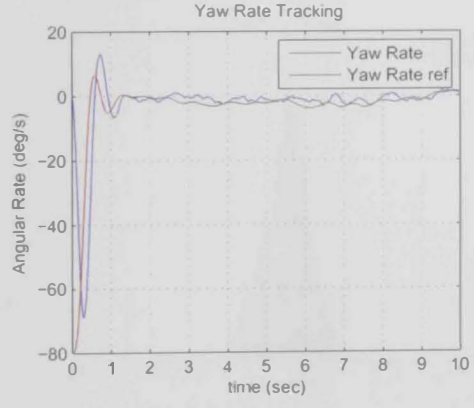
(c) Pitch rate tracking for K_{robust}



(d) Pitch rate tracking for PID

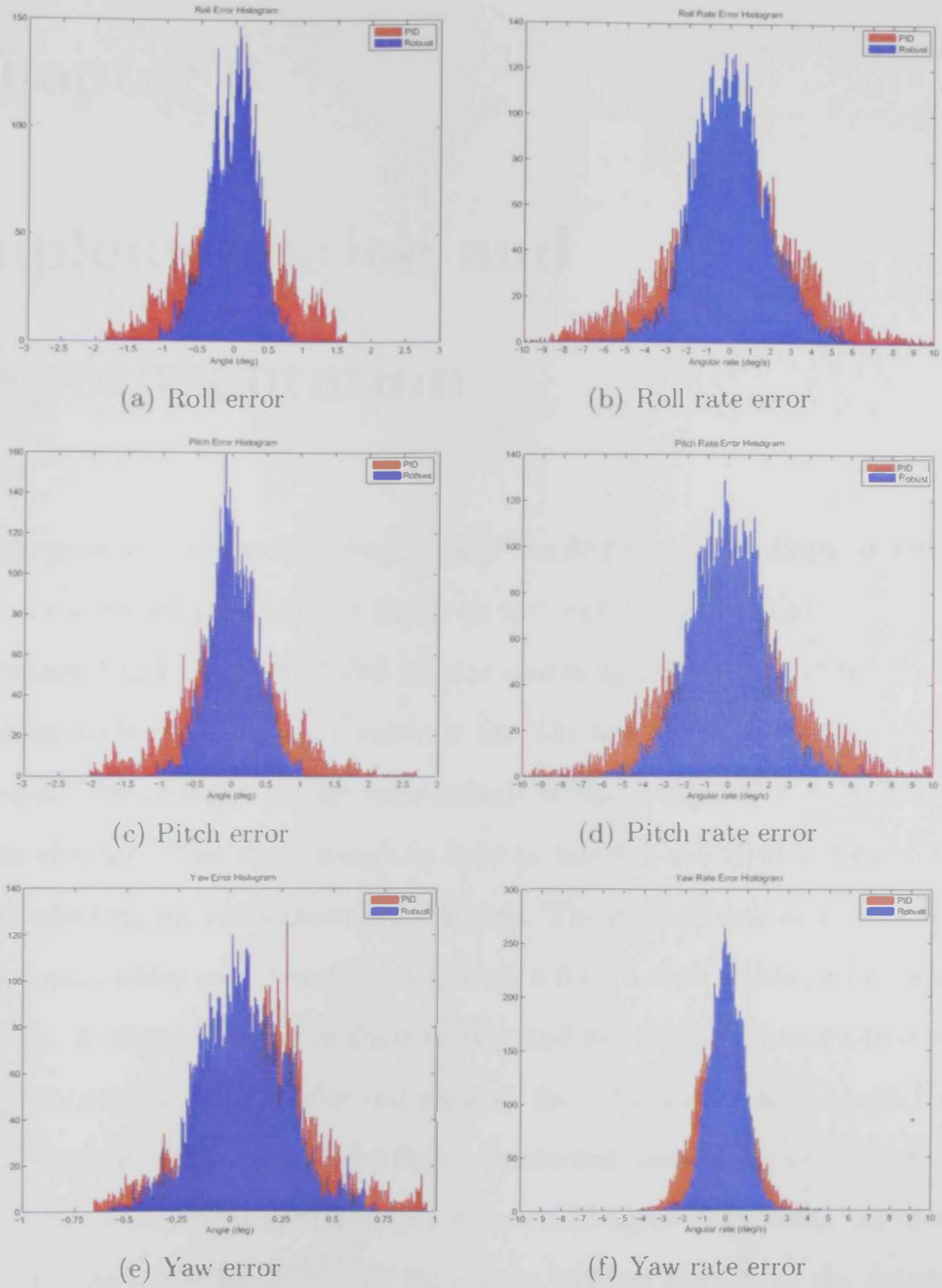


(e) Yaw rate tracking for K_{robust}



(f) Yaw rate tracking for PID

Figure 5.14: Robust controller vs. PID controller - body rate tracking comparison



(a) Roll error

(b) Roll rate error

(c) Pitch error

(d) Pitch rate error

(e) Yaw error

(f) Yaw rate error

Figure 5.15: Histogram comparison of the error signal for the case of PID (red) controller and Robust controller (blue).

Chapter 6

Implementation and Experimentation

The purpose of this chapter is present how the ArduCopter platform was utilized to conducted experimental flights on the controllers obtained previously in Chapter 4 and Chapter 5. The chapter covers initially details of the modifications made to ArduCopter software and the new functions added to the software. The preflight testing environment is also presented and explained in this chapter. This environment is used to test the modified software before conducting the real experimental flights. The environment is an existing environment which uses ArduCopter ground software with flightgear for visualization. A stage of extensive flight testing and fine tuning of the controllers was performed, and the results and analysis for both controllers is presented in this chapter. The experimental flights conducted covered the nominal case scenario, uncertainty in inertia, and a scenario of input disturbance injected in flight. The important subset of the results and the analysis of the data is presented in this chapter.

6.1 Software Development

In this chapter, it is explained how the controllers obtained were implemented as well as how the platform ArduCopter with its avionics kit was utilized to conduct experiments. The scope of the implementation covers only part of the inner loop controller. The implementation and experiments include roll and pitch attitude control as well as angular rate control on roll and pitch. Yaw loops and altitude loops were excluded from the implementation. The reason for excluding the yaw loop was that the original system suffers from deviation from reference heading. The system seemed unaware of this deviation, however it was noticed visually. This problem could be related to the specific version of the autopilot hardware design (APM v2.5), or it could be a manufacturing problem with the specific autopilot board used. Furthermore, the yaw model requires more effort and research in the modeling, rather than control, as shown in the model mismatch in Chapter 3. The reason for excluding the altitude loop is the fact that the existing ArduCopter does not have high rate altitude estimates. Raw measurements of altitude and its derivatives exist of the GPS, barometer and accelerometers. However, studying, designing, and implementing an estimator at 100 Hz require much more effort and time, which was not available within the scope of this project. Consequently, the focus of the implementation was limited to the attitude part of the inner loop.

The implementation in this research was based on release "V 2.6" of Arducopter software. This software release was used as a starting point to make the necessary modifications to the existing system to make it suitable for the research project. The development strategy followed was to utilize as much as possible from the existing functionality and facilities of the system to avoid unnecessary modifications. The two main facilities that were utilized were:

- Switching between flight modes in flight by the operator; and
- Tuning controller gains from ground software without recompiling the

source code.

The development plan was to keep an original flight mode with absolutely no modification (original source code, original gains) and modify other flight modes to activate different controllers and loops. The original flight mode was used as the primary safe mode for take-off and landing, as well as taking over when the experimental flight mode became unstable. The experimental flight mode re-used some of the existing parameters that could be modified by the ground software.[1] This was done and executed safely by ensuring that the existing functions did not conflict with the newly developed functions.

The software development process involved modifying a set of existing functions, as well as adding new functions to the system. The three main functions that were added to the system are actuation, LTI implementation in state-space form, and basic matrix operations. Since the designed controllers generate commands in units of torque, it is necessary to transform these control signals into Ardu-specific formats. This transformation was implemented in two functions, see Appendix E. First, the commanded torque was transformed into the corresponding PWM signal through the function `u2pwmFunc(float *u, float *y)`. The output of this function was then fed to `pwm2mxFunc(float *u, float *y)` to convert the PWM signal into a Ardu-specific command. These two functions then implement the actuation support for the controllers. Aside from the actuation functions, software development was done to implement an LTI system in state-space form and propagate its states over time. The propagation over time (integration) was done using the Euler discrete integration method. As the LTI system contains a set of matrices, supporting functions were also developed to implement basic matrix operations, see Appendix E for the source code:

1. `MatAdd`, for matrix addition;

[1]The existing PID parameters for lateral-loiter PID were used as gains for the experimental pitch-rate PID controller.

2. `MatMult`, for multiplying two matrices together or multiplying a scalar with a matrix; and

3. `MatSet`, for setting all matrix elements to a certain value.

The released software from the ArduCopter team was modified in many ways. The major modification made to the software for this project involved the control library, flight modes switching and functions, and logging files. The original control library is based on fixed-point operations. The modification to the library was done to include floating point operations in the controllers. The existing flight modes and switching between them was also modified. The names of the modified flight modes do not correspond with their original operations anymore and one should be aware of this when activating these flight modes. The flight mode STABILIZE was kept without modifications to be used as a backup. In Table 6.1, a list of modified flight modes is presented with the controllers and active loops for each mode.

Enum.	Name	Active Loop	Controller
0	STABILIZE	Angle	Original
1	ACRO	Angle	Exp. PID
2	ALT_HOLD	Body Rate	Exp. PID
3	AUTO	Body Rate	Exp. Robust
4	Circle	Angle	Exp. Robust

Table 6.1: Flight modes and inner-loop configuration for each mode

The column "Enum." presents the enumeration code assigned to the flight mode in the log files. The column "Name" gives the name of the flight mode in the ground software used with the system, and the columns "Active Loop" and "Controller" shows the loop that is active and commanded by RC, and the controller activated within flight mode.

The logging files were modified earlier in the research stages for identification purposes. The modifications were to log a set of chosen variables/measurements synchronously at 50 Hz. These variables included reference signals (RC

command), a control signal, body angular rates (gyroscope measurements), attitude estimation, and the controllers integrator state. The logged data were stored on-board on a flash and were downloaded after every flight in text format. Separate software was developed to retrieve the text formatted log files and to extract and manipulate the data to produce a ".csv" file suitable for analysis. The list in Table 6.2 shows the list of logged data with the corresponding units in the ".csv" files.

No.	Name	Unites	Description
1	time	sec	Current time relative to start of log
2	cmdIn1	deg	Reference Roll angle (from RC)
3	cmdIn2	deg	Reference Pitch angle (from RC)
4	cmdIn3	%	Thrust percentage (from RC)
5	cmdIn4	—	Yaw reference Not used-.
6	motorOut1	$10^{-6}sec$	Pulse width of the PWM signal for motor 1
7	motorOut2	$10^{-6}sec$	Pulse width of the PWM signal for motor 2
8	motorOut3	$10^{-6}sec$	Pulse width of the PWM signal for motor 3
9	motorOut4	$10^{-6}sec$	Pulse width of the PWM signal for motor 4
10	ahrsroll	deg	Estimated roll angle
11	ahrspitch	deg	Estimated pitch angle
12	ahrsyaw	deg	Estimated yaw angle
13	gyrox	deg/sec	Filtered gyro measurement on x-axis
14	gyroy	deg/sec	Filtered gyro measurement on y-axis
15	gyroz	deg/sec	Filtered gyro measurement on z-axis
16	int_RR	deg/sec^2	Integrator state of Roll-Rate controller
17	int_PR	deg/sec^2	Integrator state of Pitch-Rate controller
18	mode	N/A	Active flight mode at the time

Table 6.2: Log files structure description

6.2 Testing Environment

The purpose of the testing environment described in this section is to ensure the correct functionality of the developed source code and resolve issues and bugs that presented earlier in a managed environment. The testing environment presented is not concerned with the dynamical response of the closed-loop system. The reason for this is that the dynamical response of the closed-loop

system has already been checked and verified against a known model (refer to the full non-linear model in Chapter 2). Furthermore, the system model provided by the ArduCopter development team (which is used in this environment) does not match the system used in this research. For these two reasons, the dynamical response of the closed-loop system is not analyzed and the focus is kept on verifying the correct functionality of the developed and modified source code (ie. actuation subsystem, state-space controller propagation, logics).

The testing entailed two steps. The first step was writing the S-functions, then integrating the written code onto the Ardu-pilot mega (APM) board (autopilot board) for testing within the hardware-in-the-loop simulation (HILS) environment. In the first step, the blocks developed in the Simulink environment were implemented in C programming language and incrementally tested with the rest of the system simulation. This was applied to the actuation functions and discrete implementation of the robust controller.

After testing and verifying the functions developed in the Simulink environment, the source code was taken into the integration stage and into the complete system code of the ArduCopter. The modifications to the original system were also done at this stage and the entire system code (with the added functions) was compiled and flashed onto the autopilot board. Then, the autopilot in its current state was tested in the HILS environment presented in Figure 6.1.

As can be seen from Figure 6.1, the HILS environment consists of the following components:

- An APM board;
- An RC transmitter and receiver;
- A personal computer (PC) with ground station software (Mission Planner); and

- A flight simulator (FlightGear is one of the options).

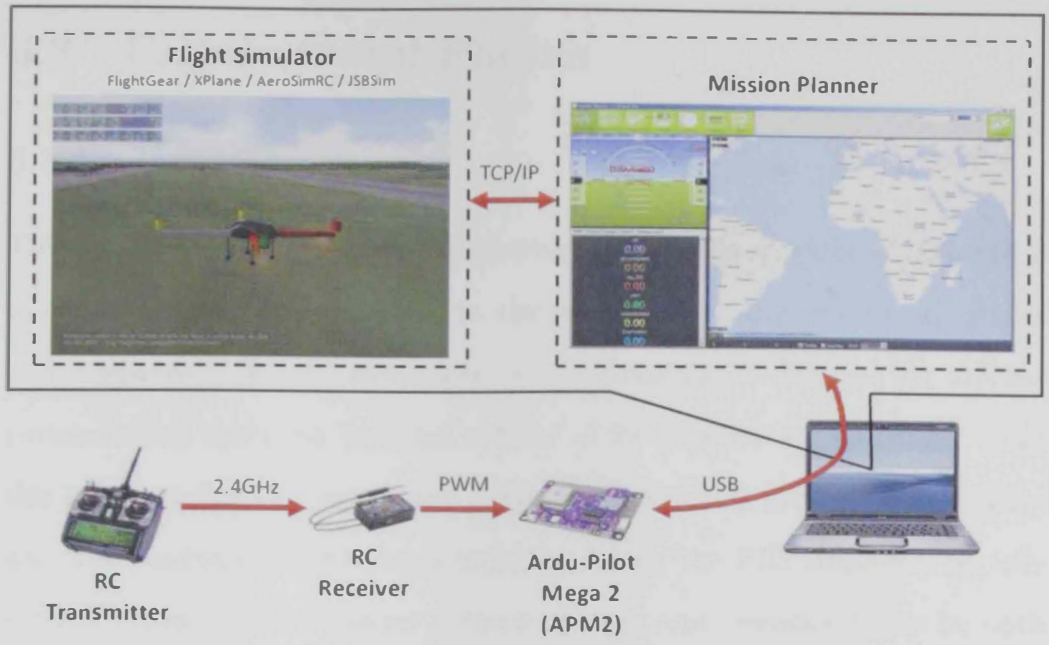


Figure 6.1: Hardware-in-the-loop-simulation environment setup

Testing in the HILS environment presented in Figure 6.1 was the last step before conducting the experimental flights with the real platform. The controller in this setup is the APM board with the software running while the model is simulated in the flight simulator. The APM board receives PWM signals from the RC receiver that are interpreted as reference for the controllers. The PWM signals are driven by the human pilot through the RC transmitter. The controllers in the APM then produce the appropriate motor commands that should be forwarded to the motors. The ground software then reads the motor commands through the USB channel and forwards these commands to the motors of the simulated aircraft within the flight simulator. The simulated aircraft propagates accordingly and the flight simulator produces simulated measurements for the various sensors (GPS, barometer, gyroscope, accelerometers, etc.). The ground software (Mission Planner) then forwards these measurements as raw values to the APM board. This cycle of data flow continues during the simulation run and the system which is run with the

newly developed software is verified and checked.

6.3 Experimental Flights

6.3.1 Overview

This section aims at presenting, discussing, and analyzing the results obtained from the conducted experiments in this study. Moreover, the testing stages and hardware added for robustness test are explained. The testing and experimentation activities included a total of 29 recorded experiments. From this large number of experiments data, 14 experiments are usable for analysis. The conducted experiments included tuning the PID controller (mostly indoors), nominal test cases for both controllers, and robustness tests for both controllers. This section presents only a subset of the most relevant results.

The experiments were conducted over two stages, namely the nominal tests and, after that, the robustness tests. In the nominal test stage, the controllers were tested in nominal flight conditions and envelope to ensure that the controllers work without major problems. Tuning the PID controller was performed during this stage. In the robustness testing stage, a number of disturbances were injected into the system in flight and the performance of both controllers was checked. In order to perform this test, the original platform was augmented with extra hardware to enable injection of a set of disturbances. The connection of the extra hardware to the original plant is shown in Figure 6.2.

Three types of disturbances were injected, namely inertia increase in roll axis, torque step disturbance, and torque noise. The torque step and torque noise disturbances are controlled by the pilot and can be engaged and disengaged on demand. During the flight, and immediately after disengaging the disturbance, a pitch reference command pattern was executed manually by the pilot. The pattern executed was a full pitch command of $\pm 45 deg$. The reason

for executing the pattern is that the autopilot cannot measure or record the time frame in which the disturbance was injected. Therefore, recognizing the time frame when the disturbance was engaged could be confused with normal system behavior. The pattern is important to be able to recognize the time frame in which disturbances were injected when analyzing the logged data.

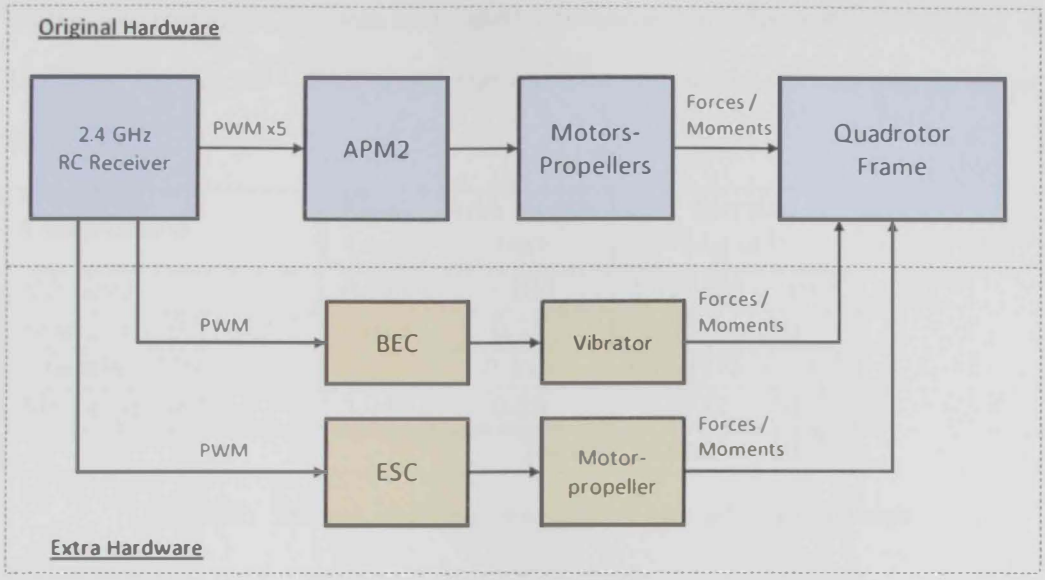


Figure 6.2: Extra hardware diagram and connections to the original system

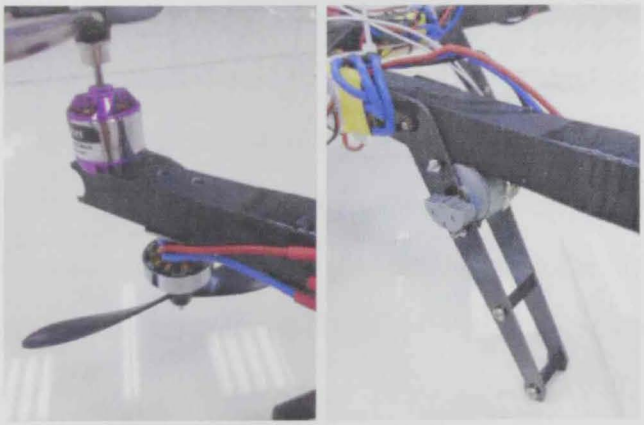
The extra hardware added to the platform for disturbances injection consists of a vibration motor and a small-scale motor propeller. The vibrator motor was used to inject a sinusoidal torque disturbance with an estimated frequency of 5 Hz, while the motor propeller was used to inject a step-torque disturbance signal. The pictures in Figure 6.3 show the vibrator motor and the small-scale motor propeller. These two additional hardware components require controller circuits, an electronic speed controller (ESC) (for brushless motors), and a brushed electronic controller (BEC) (for brushed motors) to drive the voltage line. The motor-propeller setup used previously in Chapter 3 was reused to estimate the torque produced by the extra motor propeller. The two motors were connected to separate PWM channels as shown in Figure 6.2. These two channels are controlled by a switch in the RC transmitter. A "pushbutton" type of switch was chosen to ensure that the disturbance will

engage when the switch is pressed and disengage when the switch is released.

The extra set of hardware was mounted to affect only the inertia of the roll axis. The masses of all the extra hardware as well as the arm length of the mounting point was measured and noted - see Table 6.3-. In total, the extra hardware increased the inertia on roll axis by $4.9621 \times 10^{-3} kg.m^2$. The inertia on roll axis I_{xx} was estimated previously in Chapter 3 to be $I_{xx} = 15.654 \times 10^{-3} kg.m^2$. Therefore, the increase of inertia on roll axis is about 30%.

Components	Mass (kg)	Arm length (m)	Inertia $(kg.m^2)$
Vibrator	0.033	0.185	1.129425×10^{-3}
Brushed ESC (BEC)	0.013	0.11	0.1573×10^{-3}
Brushless ESC	0.023	0.115	0.304175×10^{-3}
Motor-propeller	0.043	0.28	3.3712×10^{-3}
		Total:	4.9621×10^{-3}

Table 6.3: Masses and arm lengths for the added hardware.



(a) Motor-Propeller (b) Motor Vibrator

Figure 6.3: Extra motor propeller and vibrator mounted onto the original system for disturbances injection

6.3.2 Nominal Test

6.3.2.1 PID Tuning

During the nominal test stage, the PID gains were tuned to achieve an acceptable gains set. The tuning process took a couple of iterations and portions of the gains used is shown in Table 6.4.

The first tested gain set for the PID was the gains obtained previously in Chapter 4. An appropriate flight mode was activated to test the body angular rate alone and then the angle loop was added. The closed-loop system using gains set No. 1 suffered from steady-state error in both angular-rate loops (roll rate and pitch rate). The controller was not able to compensate for the source of this error. It was noticed that the error on the pitch axis was larger than on the roll axis. The source of this error was the imbalance in the actuators in which the motor propeller sets give different thrusts for the same command. This is justified later by inspecting the integrator state in Figure 6.6a, in which the integrators state was built to value that achieves the balance. The shift in the CG location also contributed to the steady-state error. A second set of gains was tested (No. 2) to enhance the response of the closed-loop system by increasing the gain of the controller. Then, in gain sets No. 3 and No. 4, integration action was added to handle the imbalance in the actuation and the shifted CG location. The gains in set No. 4 were selected as the final gains by the research team.

No.	K_p	K_i	K_d
1.	10	0	1
2.	15	0	1.5
3.	15	0.5	1.5
4.	15	1	1.5

Table 6.4: Sets of PID gains used during the tuning process

The results presented in Figure 6.4 are for angular rate tracking for gain set No. 4. An appropriate flight mode was activated to accept RC commands as

body angular rate commands and activate the PID controller for angular rate control. The plots show that the rate loop works well, with no steady-state error. It can also be noticed that there was a small overshoot in the system output upon reaching the angular rate command.

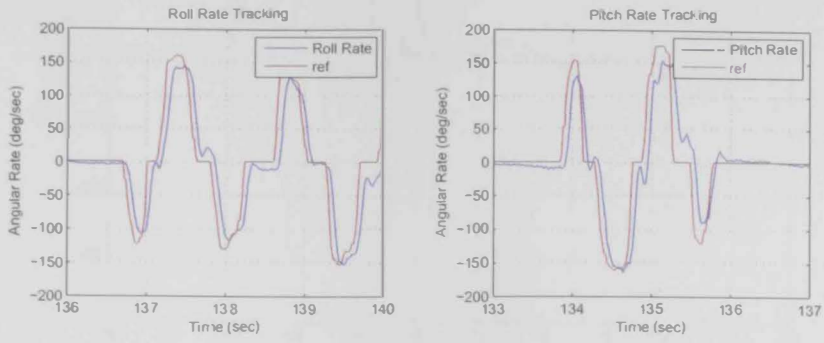


Figure 6.4: Roll-rate and pitch-rate tracking for gain set No. 4

After achieving acceptable gains in the angular-rate loop, the attitude loop was closed and tested. The results presented in Figure 6.5 is for the system response to pitch attitude command. The plot shows that the closed-loop system took one second to recover from 25 degrees to 0 degrees. This response is considered acceptable and it conforms to the previous design requirement of a one second time response for the attitude loop.

The controller integrator state, commanded moments, and PWM commands are plotted and presented in Figure 6.6 for the same time frame as in the pitch-tracking plot. The moment plot shows that the control signal was slightly noisy and that the actuators were saturated for short periods. It can also be noticed that the saturation occurred at the time instances when the reference changed. This saturation occurred due to the derivative action when the reference command changed dramatically.

The integrator action helped to overcome the steady-state error problem when the state of the integrator settled to a trim point. This design seems to work well when the closed-loop system is at a fixed point. However, when the closed-loop system is excited, the integrator state changes and the closed-loop system loses trim when the reference command is brought back to zero.

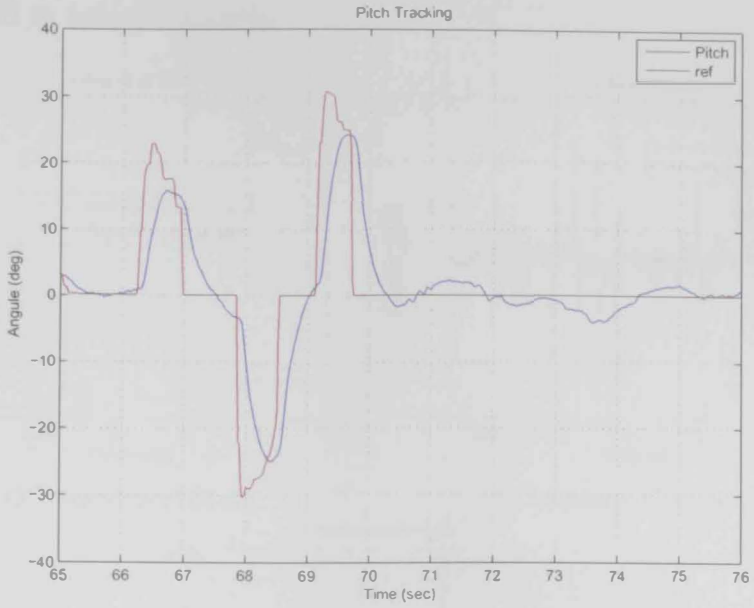
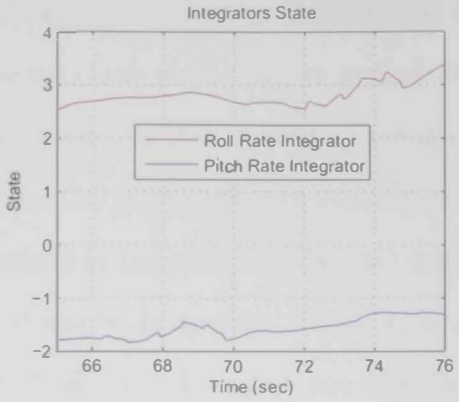


Figure 6.5: System response to commanded pitch attitude

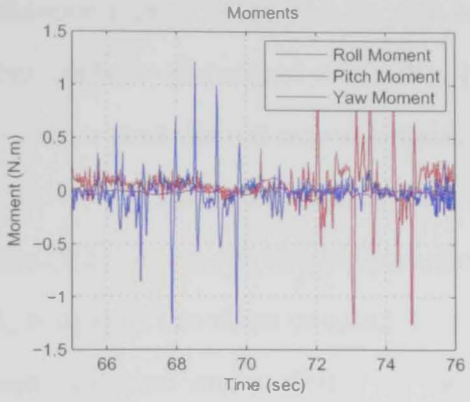
The reason for this is that the integrator is always building up its state and contributing to the total action during reference tracking. This causes a steady-state error with slow recovery. The plots in Appendix F show a time frame from the same experiment when the steady-state error returned and recovered slowly.

There are two possible solutions for overcoming the integrator state problem. One solution is to increase the integration. This will make the recovery from the steady-state error much faster. However, increasing the gain of the integrator affects the closed-loop systems response and requires redesigning the controller to use a complete PID gain set instead of PID with dominant proportional and derivative actions and minor integration action. An alternative solution to the problem is to design a non-linear integrator, which will activate integration mainly during a zero condition and deactivate during reference tracking. The reason for using the non-linear integrator is to allow the proportional and derivative actions of the controller to be responsible for the tracking command while keeping the integrator action responsible for building a state to achieve a trim point. The two solutions could be tested and

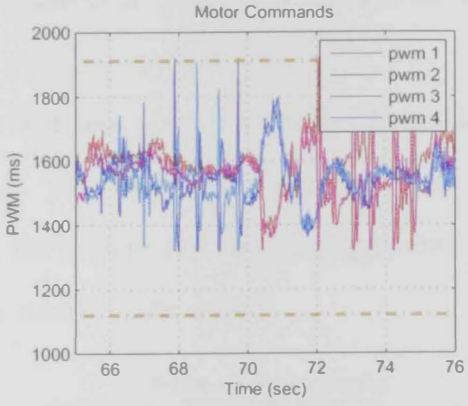
investigated in future research.



(a) Integrators' State



(b) Moment Commands



(c) PWM Commands

Figure 6.6: Controllers' integrator state and control signals plots

6.3.2.2 Robust Controller

The experiments conducted on the robust controller were initially with the original platform, without the extra hardware. The data presented in this section are for the platform with nominal inertia on pitch axis and uncertain inertia on roll axis. The analysis was done on the pitch axis, since it is considered to be the nominal axis. First, the testing was done with only the rate loop. After gaining enough confidence in the rate loop, the attitude loop was closed and the nominal test for the system was completed.

The results for the test on the body angular rate loop for the robust controller are shown in Figure 6.7 and the plots for closing the attitude loop are

shown in Figure 6.8. As presented, the robust controller works very well in stabilizing the system and tracking a reference signal. It can also be seen from the rate loop plots that the overshoot also exists for the robust controller. Furthermore, the rate loop plots in Figure 6.7 show that the roll rate was disturbed when the pitch rate was excited, and vice versa. The same phenomenon was noticed in the plots for the PID controller. The reason for this phenomenon is that the body angular rates are, in fact, coupled. The cross coupling between the body angular rates has already been discussed and included in the full non-linear model in Chapter 2. This result verifies the cross coupling included in the model.

The robust controller was able to stabilize the pitch angle as well as track a reference command. The presented data show that the closed-loop system was able to stabilize the system and track exaggerated reference signals within 0.5 seconds, with no steady-state error.

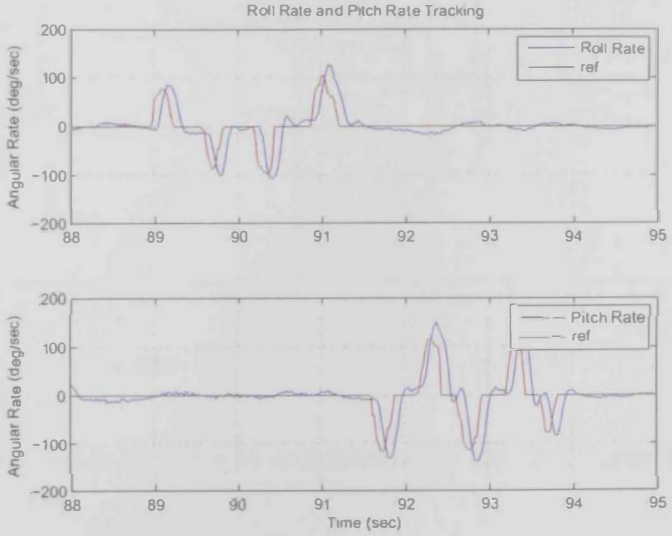


Figure 6.7: Angular-rate tracking plots for the robust controller

The plots shown in Figure 6.9 are for the actuation of the system during the same time frame as in Figure 6.8. The moments and PWM commands for the robust controller are not saturated as much as the PID controller. Moreover, the noise level of the control signal was very little compared to the noise level

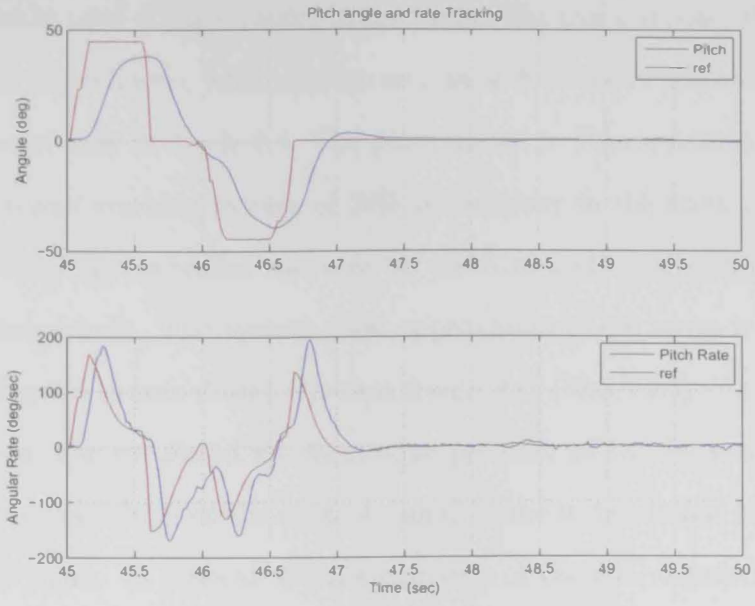
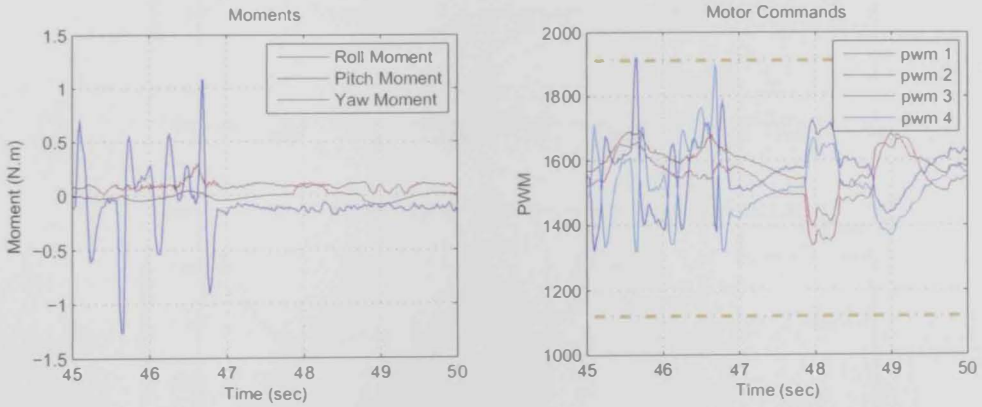


Figure 6.8: Pitch tracking plots for the robust controller

of the PID controller.



(a) Moments

(b) PWM commands

Figure 6.9: Moments and PWM commands for the same time frame as in the pitch tracking plot

6.3.3 Robustness Test

6.3.3.1 PID Performance

The PID controller was tested for robustness and the results are presented in this sub-section. The first robustness test was to check the controllers

performance in case of uncertainty in the plant. For this purpose, the analysis was done on the roll axis, which has an inertia of 30% above the nominal value as shown previously in Table 6.3. The plots shown in Figure 6.10 is for system stabilization and tracking in case of 30% uncertainty in the inertia. The data show that the PID controller was able to stabilize and successfully track roll reference commands. The system took approximately 0.6 seconds to recover from -25 degrees to zero during the time frame, $t \subseteq [73.6, 74.2]$. The actuation in the system was saturated for short time periods, as can be seen in Figure 6.11, and the noise level of the control signal seems to be similar than that of the nominal plant. In general, the performance of the PID controller was not affected dramatically by the 30% uncertainty.

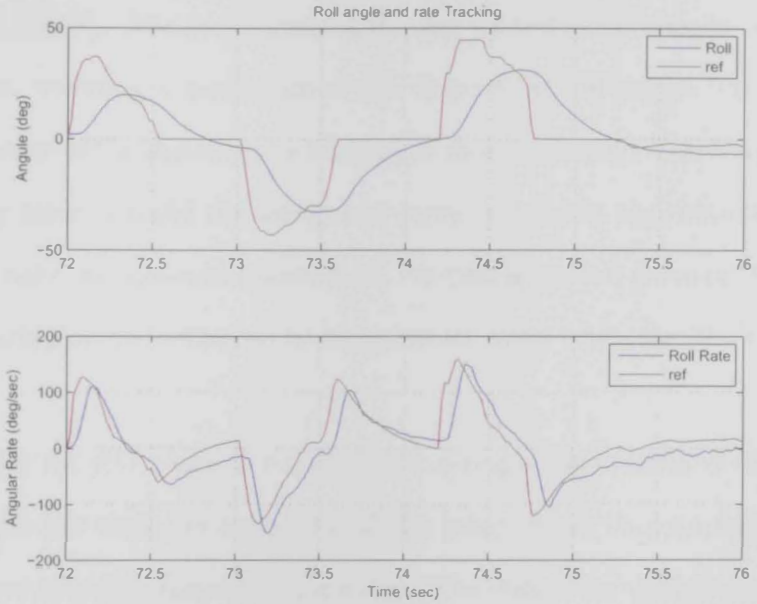
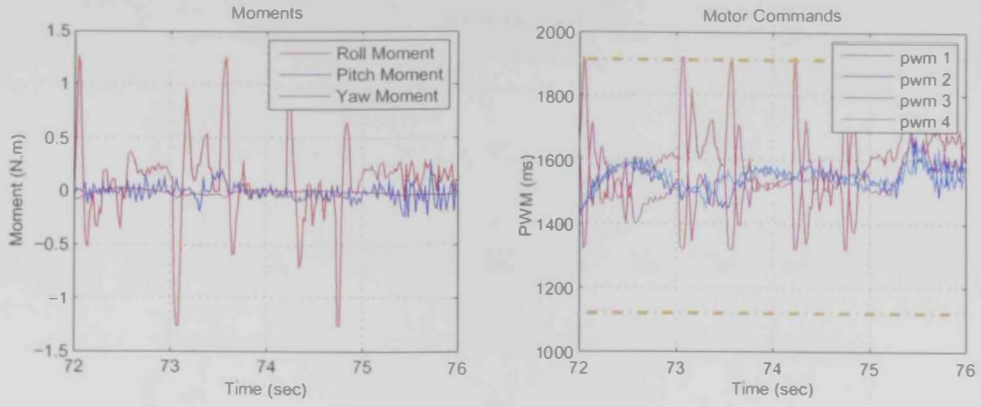


Figure 6.10: Roll and roll-rate tracking for the PID controller in case of 30% uncertainty in inertia

After testing the PID controller in case of uncertainty, the performance of the controller was checked in the presence of disturbances. The disturbances injected were the sinusoidal torque disturbance from the vibrator and the step torque disturbance from the extra motor-propeller. Unfortunately, the vibration motor was not strong enough to affect the system.

The plots shown in Figure 6.12 are for roll stabilization and tracking in



(a) Moments (b) PWM commands

Figure 6.11: Moments and PWM commands for roll tracking in case of 30% uncertainty

the presence of disturbance. The disturbance was injected during the time frame [221, 224.5]. The error caused by this disturbance was as wide as the disturbance, reaching a maximum of 8.5 degrees on roll angle. The controller in the presence of the disturbance seemed to recover slowly. The PID controller took a long time to build the integrator state and reject the disturbance. The integrator state, as shown in Figure 6.13, reached a certain value in the presence of the disturbance and returned to its previous value when the disturbance was removed.

Although the PD controller with slight integration combined the good response of the PD with the auto-trim of the integration, the controller was not able to reject the disturbance fast enough. The controller is not suitable if disturbances with similar characteristics exist (such as wind gusts in an outdoor environment).

6.3.3.2 Robust Controller Performance

Similar to how the PID controller was tested for robustness, the robust controller was also tested in two steps. Initially, the controller was tested for a plant with 30% uncertainty and then the controller was tested for disturbance rejection.

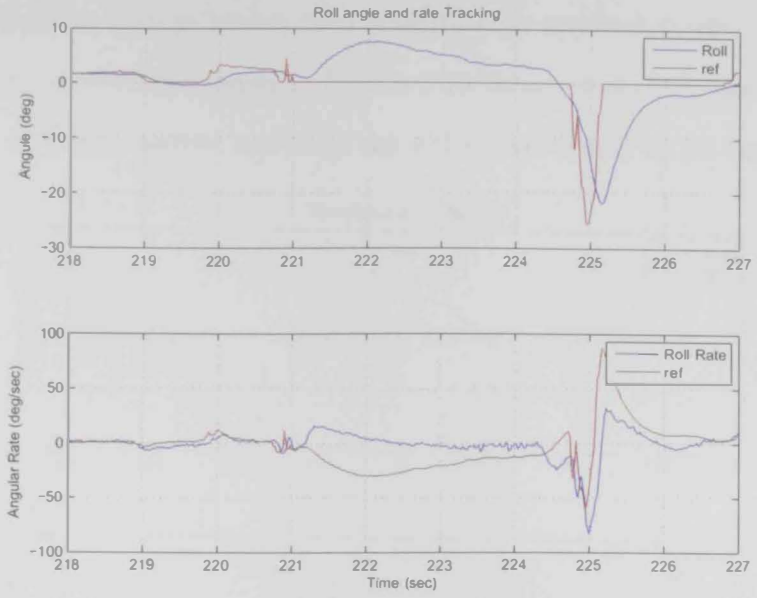


Figure 6.12: Roll and roll-rate tracking with an injected step disturbance signal for the PID controller

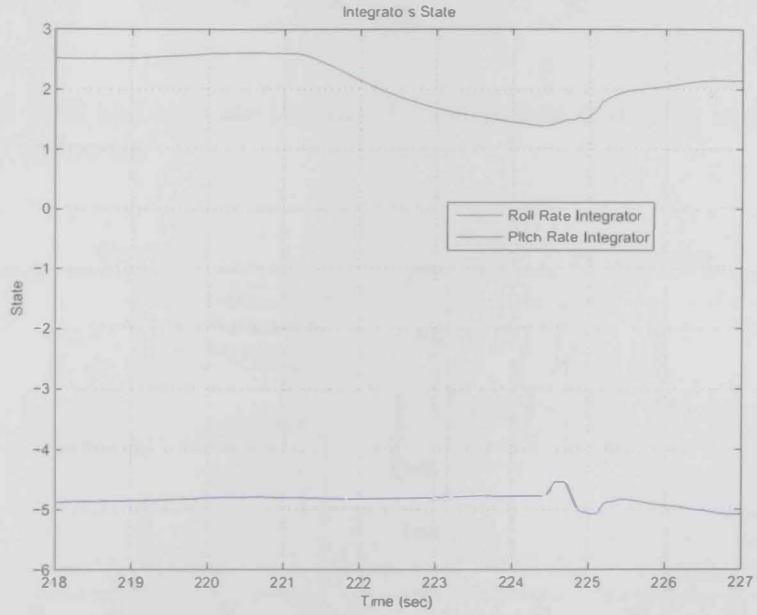


Figure 6.13: Integrator's state in the presence of a step disturbance signal

The data presented in Figure 6.14 are for roll and roll-rate tracking in case of uncertain inertia. The plots show that the robust controller is capable of good stabilization and tracking of the reference command. The performance seems to be similar to the nominal case. Although the system attitude was relatively large and considered aggressive ($\approx$ 45 degrees), the controller was

able to bring the system into a zero condition in approximately one second. Furthermore, the control signal in Figure 6.15 shows that there was almost no saturation in the actuation and that the noise level seems to be very small.

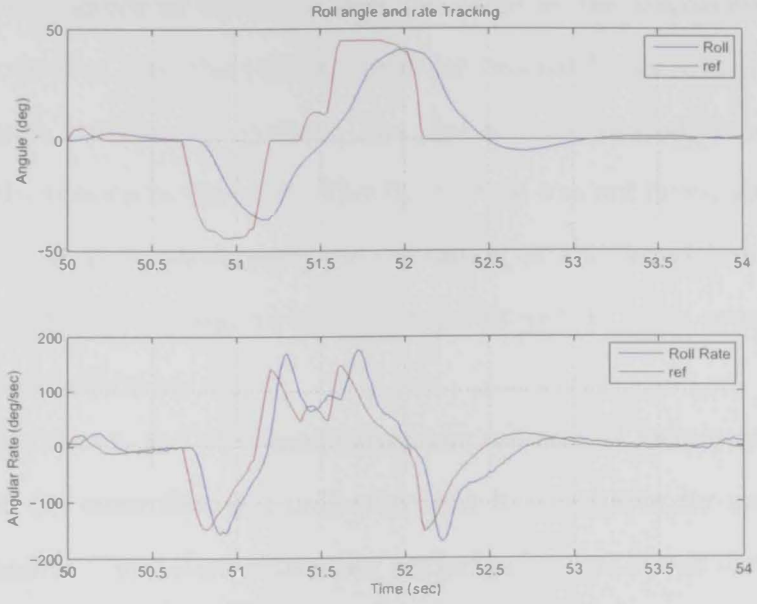
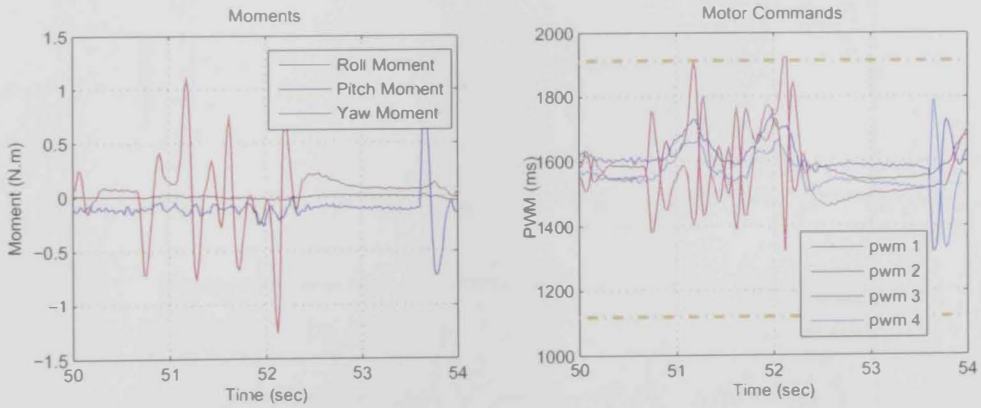


Figure 6.14: Roll and roll-rate tracking for the robust controller in case of 30% uncertainty in inertia



(a) Moments

(b) PWM commands

Figure 6.15: Moments and PWM commands for roll tracking with 30% uncertainty

The robust controller was also tested for disturbance rejection. The plot in Figure 6.16 shows the roll and roll-rate tracking in the presence of step disturbance. This test was repeated three times in the same experiment (only

one sample out of the three is shown in the figure) and the disturbance was injected for three seconds. The error in the tracking according to the plot had a duration of one second and reached a maximum of 8.5 degrees. The maximum magnitude of the error was the same as the maximum error for the PID controller, but the robust controller reacted faster and rejected the disturbance to stabilize the system. The plot also shows that there was high frequency fluctuation in the error. This fluctuation was not investigated during this research as it requires good understanding of the aerodynamics of two overlapping propellers (original system propeller and an extra propeller) that have different airflow.

The experiments and the results from the robustness tests proved the robustness of the controller and indicated that it is suitable for the rejection of disturbances. The robust controller performed much better than the PID controller in these tests and is able to handle better changing platform gains as well as trim points.

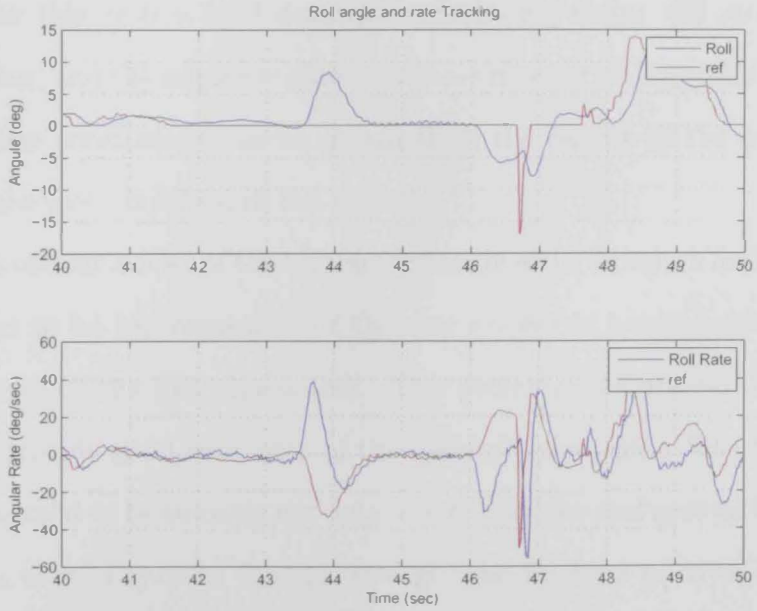


Figure 6.16: Roll and roll-rate tracking with an injected step-disturbance signal for robust control

Chapter 7

Conclusions

During the research presented in this thesis, a large number of outcomes were reached and results obtained by implementing a robust controller for a quad rotor system and comparing this designed robust controller with a PID controller. After analysis and testing, a number of conclusions can be drawn which will add value to the field of applied quad-rotor control. After investigating existing research on the topic, the inner-loop control for a quad-rotor system was successfully designed, implemented and applied. Then, robust control was achieved for this system by following the following steps: (1) modeling, (2) identification, and (3) implementation of the controller on an ArduCopter system. Various conclusions can be drawn from the results of the experiments and from the steps involved in the process.

The non-linear model of the system was realized in Simulink and the model was verified to be representative of the real system in hover condition (with the exception of the yaw dynamics). The presented identification methods proved to provide good estimates of the system parameters. The final model obtained seemed to be suitable not only to simulate the real system but also for designing a control system. Further research can be done to cover more flight envelopes and to study and understand the system dynamics in high-speed flights.

In this research, the modified ArduCopter system proved to be an environment suitable for R&D. The complete environment, which consists of the

Simulink model that represents the platform and the ArduCopter environment, the HILS environment, and the actuation module, all served as good development tools to design and test control systems in a managed and safe manner. However, further work can be done to design and implement a full-state estimator in order to proceed with implementing the outer-loop controllers.

The analysis of the data in this research also presents a better understanding of yaw dynamics. During the verification of the model, the modeled yaw moment was compared to the actual moment estimated from the experiments and they were found to not match properly. The model used in this research, and various other studies, considers only the drag from the blade rotation as the dominant factor in producing torque and neglects the motor anti-torque. Upon analyzing the data of the experiments, it can be seen that the moment on yaw was caused by the motors anti-torque as the dominant contributor. This mismatch between the model and the real system makes the model a poor representative of the system in yaw dynamics and not suitable for yaw controller design. More investigation and remodeling need to be done on yaw dynamics and further identification methods have to be applied to identify or estimate the anti-torque of the motors and any other factors that may contribute to the produced torque by the system.

From a control perspective, this research presents results and analyses of real experiments with PID and robust controllers in nominal, disturbance rejection, and cases of uncertainty. The classical PID controller was designed analytically in the rate loop and Root-Locus based in the angle loop, while the robust controller was designed using the H_∞ method and a GS/T scheme. Both controllers were designed, implemented successfully, and underwent extensive testing and experimentation.

The PID controller seemed to have a good dynamical response and successful tracking, as well as good capability of handling 30% uncertainty in the inertia. However, the controller had difficulty removing the steady-state error

and trimming the system. Another drawback of the controller is its relatively noisy actuation signal and the saturation in the control signal when the reference changes dramatically. Additionally, the controller took much longer rejecting the injected disturbances in flight.

In contrast, the robust controller seemed to have less problems and difficulties. The nominal testing showed that the robust controller is very capable of handling the system in nominal case and it performed well in terms of stabilization and tracking. Moreover, the control signal for the robust controller was not noisy and less saturation was noticed when compared to the PID controller. The robust controller also performed very well in the robustness tests. The controller seemed to be capable of handling the 30% uncertainty without sacrificing performance and very capable of rejecting the injected disturbances in flight.

More research can still be done on quad-rotor control. One of the areas to investigate is to redesign and experiment with the PID controller (with higher integration action) and analyze the performance and robustness of the controller. Another area to explore is the design of the robust controller using the extended variation of the GS/T scheme. The extended version of the scheme is supposed to ensure even better tracking for the robust controller. Another robustness test can be conducted to check the controllers performance in speed flights and identify the issues that arise in a wider flight envelope. Further, the control system could be completed to its full state by including the outer-loop controllers in the system as well as waypoint navigation.

Appendix A

The general equation for the actuation subsystem is:

$$U = \begin{bmatrix} \tau_\phi \\ \tau_\theta \\ \tau_\psi \\ T \end{bmatrix} = \begin{bmatrix} 0 & -bl & 0 & bl \\ bl & 0 & -bl & 0 \\ d & -d & d & -d \\ b & b & b & b \end{bmatrix} \begin{bmatrix} \Omega_1^2 \\ \Omega_2^2 \\ \Omega_3^2 \\ \Omega_4^2 \end{bmatrix} = M\hat{\Omega} \tag{1}$$

Inverting the model is done by inverting the actuation matrix, M , where the actuation matrix is given by:

$$M = \begin{bmatrix} 0 & -bl & 0 & bl \\ bl & 0 & -bl & 0 \\ d & -d & d & -d \\ b & b & b & b \end{bmatrix} \tag{2}$$

The inverse M^{-1} is obtained using the general form of matrix inverse:

$$M^{-1} = \frac{[cofactor(M)]^T}{|M|} \tag{3}$$

The cofactor of the matrix M is obtained as follows:

$$\begin{bmatrix}
\begin{vmatrix} 0 & -bl & 0 \\ -d & d & -d \\ b & b & b \end{vmatrix} & -\begin{vmatrix} bl & -bl & 0 \\ d & d & -d \\ b & b & b \end{vmatrix} & \begin{vmatrix} bl & 0 & 0 \\ d & -d & -d \\ b & b & b \end{vmatrix} & -\begin{vmatrix} bl & 0 & -bl \\ d & -d & d \\ b & b & b \end{vmatrix} \\
-\begin{vmatrix} -bl & 0 & bl \\ -d & d & -d \\ b & b & b \end{vmatrix} & \begin{vmatrix} 0 & 0 & bl \\ d & d & -d \\ b & b & b \end{vmatrix} & -\begin{vmatrix} 0 & -bl & bl \\ d & -d & -d \\ b & b & b \end{vmatrix} & \begin{vmatrix} 0 & -bl & 0 \\ d & -d & d \\ b & b & b \end{vmatrix} \\
\begin{vmatrix} -bl & 0 & bl \\ 0 & -bl & 0 \\ b & b & b \end{vmatrix} & -\begin{vmatrix} 0 & 0 & bl \\ bl & -bl & 0 \\ b & b & b \end{vmatrix} & \begin{vmatrix} 0 & -bl & bl \\ bl & 0 & 0 \\ b & b & b \end{vmatrix} & -\begin{vmatrix} 0 & -bl & 0 \\ bl & 0 & -bl \\ b & b & b \end{vmatrix} \\
-\begin{vmatrix} -bl & 0 & bl \\ 0 & -bl & 0 \\ -d & d & -d \end{vmatrix} & \begin{vmatrix} 0 & 0 & bl \\ bl & -bl & 0 \\ d & d & -d \end{vmatrix} & -\begin{vmatrix} 0 & -bl & bl \\ bl & 0 & 0 \\ d & -d & -d \end{vmatrix} & \begin{vmatrix} 0 & -bl & 0 \\ bl & 0 & -bl \\ d & -d & d \end{vmatrix}
\end{bmatrix} \tag{4}$$

$$= \begin{bmatrix}
-b^2ld + b^2ld & -b^2ld - b^2ld - b^2ld - b^2ld & b^2ld - b^2ld & b^2ld + b^2ld + b^2ld + b^2ld \\
b^2ld + b^2ld + b^2ld + b^2ld & -b^2ld + b^2ld & -b^2ld - b^2ld - b^2ld - b^2ld & b^2ld - b^2ld \\
b^3l^2 + b^3l^2 & -b^3l^2 - b^3l^2 & b^3l^2 + b^3l^2 & -b^3l^2 - b^3l^2 \\
b^2l^2d + b^2l^2d & b^2l^2d + b^2l^2d & b^2l^2d + b^2l^2d & b^2l^2d + b^2l^2d
\end{bmatrix} \tag{5}$$

The cofactor of matrix M is calculated as:

$$cofactor(M) = \begin{bmatrix}
0 & 4b^2ld & 2b^3l^2 & 2b^2l^2d \\
-4b^2ld & 0 & -2b^3l^2 & 2b^2l^2d \\
0 & -4b^2ld & 2b^3l^2 & 2b^2l^2d \\
4b^2ld & 0 & -2b^3l^2 & 2b^2l^2d
\end{bmatrix} \tag{6}$$

Obtaining the determinant of matrix M is also required. The determinant is obtained as follows:

$$|M| = \begin{vmatrix} 0 & -bl & 0 & bl \\ bl & 0 & -bl & 0 \\ d & -d & d & -d \\ b & b & b & b \end{vmatrix} = bl \begin{vmatrix} bl & -bl & 0 \\ d & d & -d \\ b & b & b \end{vmatrix} - bl \begin{vmatrix} bl & 0 & -bl \\ d & -d & d \\ b & b & b \end{vmatrix}$$

$$= bl(b^2ld + b^2ld + b^2ld + b^2ld) - bl(-b^2ld - b^2ld - b^2ld - b^2ld)$$

$$= bl(4b^2ld) + bl(4b^2ld)$$

$$= 4b^3l^2d + 4b^3l^2d$$

$$= 8b^3l^2d \tag{7}$$

Finally, after obtaining the cofactor and determinant of matrix M , the inverse can be obtained using the general form of matrix inversion presented earlier:

$$M^{-1} = \frac{1}{8b^3l^2d} \begin{bmatrix} 0 & 4b^2ld & 2b^3l^2 & 2b^2l^2d \\ -4b^2ld & 0 & -2b^3l^2 & 2b^2l^2d \\ 0 & -4b^2ld & 2b^3l^2 & 2b^2l^2d \\ 4b^2ld & 0 & -2b^3l^2 & 2b^2l^2d \end{bmatrix} \tag{8}$$

$$M^{-1} = \begin{bmatrix} 0 & \frac{1}{2bl} & \frac{1}{4d} & \frac{1}{4b} \\ -\frac{1}{2bl} & 0 & -\frac{1}{4d} & \frac{1}{4b} \\ 0 & -\frac{1}{2bl} & \frac{1}{4d} & \frac{1}{4b} \\ \frac{1}{2bl} & 0 & -\frac{1}{4d} & \frac{1}{4b} \end{bmatrix} \tag{9}$$

Appendix B

For a given linear system, $P(s)$, and two controllers, the first with proportional only feedback ($C_P(s)$), and the second with proportional and integral feedback ($C_{PI}(s)$):

$$P(s) = \frac{Y(s) + d_y(s)}{U(s) + d_u(s)} \tag{10}$$

$$C_P(s) = k_p \tag{11}$$

$$C_{PI}(s) = \frac{k_p s + k_i}{s} \tag{12}$$

Where $d_y(s)$ and $d_u(s)$ are output and input disturbances, respectively. Two closed-loop transfer functions, $CL_P(s)$ and $CL_{PI}(s)$, can be obtained for each of the two controllers:

$$L_P(s) = P(s)C_P(s) \tag{13}$$

$$L_{PI}(s) = P(s)C_{PI}(s) \tag{14}$$

$$CL_P = \frac{L_P(s)}{1 + L_P(s)} \tag{15}$$

$$CL_P = \frac{k_p \frac{Y(s)+d_y(s)}{U(s)+d_u(s)}}{1 + k_p \frac{Y(s)+d_y(s)}{U(s)+d_u(s)}}$$

$$= \frac{k_p(Y(s) + d_y(s))}{U(s) + d_u(s) + k_p(Y(s) + d_y(s))} \tag{16}$$

$$CL_{PI} = \frac{L_{PI}(s)}{1 + L_{PI}(s)} \tag{17}$$

$$CL_{PI} = \frac{\frac{k_p s+k_i}{s} \frac{Y(s)+d_y(s)}{U(s)+d_u(s)}}{1 + \frac{k_p s+k_i}{s} \frac{Y(s)+d_y(s)}{U(s)+d_u(s)}}$$

$$= \frac{(k_p s + k_i)(Y(s) + d_y(s))}{s(U(s) + d_u(s)) + (k_p s + k_i)(Y(s) + d_y(s))} \tag{18}$$

A closed-loop transfer function with zero steady-state error should have a unity DC gain, $(U(0), Y(0), CL(0))$. In other words, the plant output is following the input reference signal without steady-state error.

Table 1 shows a summary of systems (with/without integrals), noise cases, controllers, along with steady-state error analyses[1]:

P	d_u, d_y	C	e_{ss}
$U(0) \neq 0$	$d_u(s) = 0$, $d_y(s) = 0$	P	$CL_P(0) = \frac{k_p Y(0)}{U(0)+k_p Y(0)}$ $\neq 1$
$U(0) = 0$	$d_u(s) = 0$, $d_y(s) = 0$	P	$CL_P(0) = \frac{k_p Y(0)}{U(0)+k_p Y(0)}$ $= 1$
$U(0) = 0$	$d_u(s) \neq 0$, $d_y(s) = 0$	P	$CL_P(0) = \frac{k_p(Y(0)+d_y(s))}{U(0)+d_u(s)+k_p(Y(0)+d_y(s))}$ $\neq 1$
$U(0) \neq 0$	$d_u(s) \neq 0$, $d_y(s) \neq 0$	PI	$CL_P(0) = \frac{(k_p s+k_i)(Y(0)+d_y(s))}{s(U(0)+d_u(s))+(k_p s+k_i)(Y(0)+d_y(s))}$ $= 1$

Table 1: Steady-state error analyses for systems with feedback controllers.

The first scenario is for a general system that has no integrator and where all disturbances are ignored. In this case, the DC gain of the closed-loop transfer function is not a unity (a steady state error exists). This indicates that the controller is not able to achieve a zero steady state error. In the second case,

[1]Steady state error or DC gain is obtained by substituting for $s = 0$

a system is considered with an internal integrator and no disturbances. The closed loop for this case has zero steady-state error. However, when the input disturbance is added, such as in the third case, the feedback controller is unable to achieve a zero steady-state error. The last case considers systems in general (without an internal integrator) with input disturbances. The controller in the last case is capable of achieving a zero steady-state error regardless of the presence of disturbances and a plant integrator.

Appendix C

```
clear all
close all
clc

debugging_mode      = 0;
select_scheme       = 2;            % 1: S/KS/T,   2: GS/T
plot_flag           = 1;

%%
%------------------------------------------------
%|             Definitions                      |
%------------------------------------------------

Ws_Std  = @(Ms, ws, As)  (tf([1/Ms ws], [1 ws*As]))
Wt_Std  = @(Mt, wt, At)  (tf([1 wt/Mt], [At wt]))

%%
%------------------------------------------------
%|             Initializations                  |
%------------------------------------------------

s = tf('s');

tau = 0.1;
Gp = 1/(s*(tau*s+1))

switch(select_scheme)
    case 1                  % S/KS/T
        Ms=10^(6/20);        ws=4;       As=10^(-40/20);
        Mt=10^(6/20);        wt=10;      At=10^(-40/20);
        Ws = Ws_Std(Ms, ws, As);         % Shapes S
        Wt = Wt_Std(Mt, wt, At);         % Shapes T
        Wu = 0.0;

    case 2                 % GS/T
        Ms=10^(10/20);        ws=2*pi*2;      As=10^(-40/20);
        Mt=10^(10/20);        wt=2*pi*8;      At=10^(-40/20);
        Ws = Ws_Std(Ms, ws, As);         % Shapes S
        Wt = Wt_Std(Mt, wt, At);         % Shapes T
        Wu = 0.005;

    otherwise
        error('No selection of Scheme!');
end
```

```
switch(select_scheme)
    case 1                  % S/KS/T
        We_kst = Ws;
        Wu_kst = Wu
        Wy_kst = Wt

    case 2                  % GS/T
        Wy_gst = Ws;
        Wd_gst = Wu;
        Wu_gst = Wt;

    otherwise
        error('No selection of Scheme!');
end

%%
%-------------------------------------------------
%|       Setting up Augmented Plant              |
%-------------------------------------------------

switch select_scheme
    case 1
        AugG = SKST_Scheme(Gp, We_kst, Wu_kst, Wy_kst);

    case 2
        AugG  = GST_Scheme(Gp, Wy_gst, Wd_gst, Wu_gst);

    otherwise
        AugG = augw(Gp, We_skst, Wu_skst, Wy_skst);
end

%%
%-------------------------------------------------
%|           Obtaining Controller                |
%-------------------------------------------------

C0      = ltiblock.ss('GST_K',4,1,1);
options = hinfstructOptions('Display','iter')

[K, gamma, info] = hinfstruct(AugG,C0,options);
[K,CL,GAM,INFO] = hinfsyn(AugG,1,1);

%%
%-------------------------------------------------
%|             Analysis                          |
%-------------------------------------------------

L = K * Gp;

S = 1/(1+L);
T = L/(1+L);

w1 = 10^-3;
w2 = 10^4;

db_high = 30;
db_low  = -60;
```

```
104 e = eig(K);
105
106 msg = sprintf('Controller Eigenvalues:\n');
107
108 for i=1:length(e)
109     msg = [msg sprintf('\t%f\n',e(i))];
110 end
111 disp(msg);
112
113 if plot_flag==1
114
115     Ts = 0.01;
116
117     u   = ones(5/Ts,1);
118     y   = zeros(size(u));
119     dx  = zeros(order(T),1);
120     x   = zeros(size(dx));
121
122     for i=2:length(u)
123         dx   = T.a * x + T.b*u(i);
124         y(i) = T.c * x + T.d*u(i);
125         x = x + Ts*dx;
126     end
127
128     fig_h       = [];
129     fig_fname   = [];
130
131     fig_h = [fig_h figure];
132     fig_fname = [fig_fname ;   mat2cell('
          continuous_vs_desccrete')];
133         hold on;
134         plot((1:(5/Ts))*Ts,y,'r');
135         step(T,5,'b');
136         grid on;
137         title('Continuous and Descrete Controller Simulation');
138         legend('Discrete','Continuous');
139         ylim([-0.5 2]);
140
141     fig_h = [fig_h figure];
142     fig_fname = [fig_fname ;   mat2cell('all_singular')];
143         sigma(S,'b');       hold on;
144         sigma(GAM/Ws,'-.b');
145         sigma(T,'r');
146         sigma(GAM/Wt,'-.r');
147         grid on;
148         title('S & 1/Ws, and T & 1/Wt Singular Plots');
149         legend('S', '1/Ws', 'T', '1/Wt');
150         xlim([w1 w2]);
151         ylim([db_low db_high]);
152
153     fig_h = [fig_h figure];
154     fig_fname = [fig_fname ;   mat2cell('weighting_functions')
          ];
155         hold on;
156         sigma(Ws,'r');
157         sigma(Wt,'b');
158         xlim([w1 w2]);
159         grid on;
```

```
160         title('Ws and Wt Singular Plots');
161         legend('Ws','Wt');
162
163     for i = 1:length(fig_h)
164         set(fig_h(i),'Position',[100 100 600 400]);
165         saveas(fig_h(i), ['./Controllers/'  cell2mat(fig_fname(
                i))   '.fig']);
166         saveas(fig_h(i), ['./Controllers/'  cell2mat(fig_fname(
                i))   '.png']);
167         saveas(fig_h(i), ['./Controllers/'  cell2mat(fig_fname(
                i))   '.eps'],'psc2');
168     end
169 end
170
171 myK = K;
172
173 %% Design Capturing in '.mat' file
174
175 save('controller.mat','K');
176
177 switch(select_scheme)
178     case 1                  % S/KS/T
179         save('design.mat','K','GAM','Gp','Ms','ws','As','Mt','
                wt','At','Ws','Wu','Wt','We_kst','Wu_kst','Wy_kst','
                AugG','select_scheme');
180
181     case 2                 % GS/T
182         save('design.mat','K','GAM','Gp','Ms','ws','As','Mt','
                wt','At','Ws','Wu','Wt','Wu_gst','Wd_gst','Wy_gst','
                AugG','select_scheme');
183
184     otherwise
185         error('No selection of Scheme!');
186 end
```

Appendix D

```
%  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%    Wy :    Sensitivity weighting (tf,ss)                %
%    Wd :    Input weighting (static)                     %
%    Wu :    Complementary Sensitivity weighting (tf,ss)  %
%  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

function [AugG] = GST_Scheme(Gp, Wy, Wd, Wu)

valid_inputs = 1;

if(isa(Gp,'tf')==0 && isa(Gp,'ss')==0 )  valid_inputs = 0; end
if(isa(Wd,'double')==0)                   valid_inputs = 0; end

if(valid_inputs == 0)
    error(sprintf(['Invalid Inputs...\n',...
          'Inputs should be as follows:\n',...
          '     - Gp:  dynamical system          (tf,ss)\n',...
          '     - Wu:  Bounded Low-Pass Filter   (tf,ss)\n',...
          '     - Wd:  static value              (double)\n',...
          '     - Wy:  Bounded High-Pass Filter  (tf,ss)']));
else

    [Ap Bp Cp Dp] = tf2ss( cell2mat(Gp.num) , cell2mat(Gp.den) );
    [Au Bu Cu Du] = tf2ss( cell2mat(Wu.num) , cell2mat(Wu.den) );
    [Ay By Cy Dy] = tf2ss( cell2mat(Wy.num) , cell2mat(Wy.den) );

    Ad = 0;    Bd = 0;    Cd = 0;    Dd = Wd;

    np = size(Ap,1);        mp = size(Bp,2);      pp = size(Cp,1);
    nu = size(Au,1);        mu = size(Bu,2);      pu = size(Cu,1);
```

```
32      ny = size(Ay,1);          my = size(By,2);      py = size(Cy
            ,1);
33      nd = size(Ad,1);          md = size(Bd,2);      pd = size(Cd
            ,1);
34
35      A =    [ Au                 zeros(nu,ny)       zeros(nu,np)
            ;
36               zeros(ny,nu)       Ay                 By*Cp
                    ;
37               zeros(np,nu)       zeros(np,ny)       Ap
                    ];
38
39      B1 =   [ zeros(nu,mp)       zeros(nu,md)       ;
40               zeros(ny,mp)       zeros(ny,md)       ;
41               Bp                 zeros(np,md)       ];
42
43      B2 =   [ Bu                 ;
44               zeros(ny,mu)       ;
45               Bp                 ];
46
47      C1 =   [ Cu                 zeros(pu,ny)       zeros(pu,np)    ;
48               zeros(py,nu)       Cy                 Dy*Cp
                    ];
49
50      C2 =   [ zeros(pp,nu)       zeros(pp,ny)       -Cp
            ];
51
52      D11 = [ zeros(pu,mp)        zeros(pu,md)    ;
53              zeros(py,mp)        zeros(py,md)    ];
54
55      D12 = [ Du                ;
56              zeros(py,mu)      ];
57
58      D21 = [ zeros(pp,mp)       Dd      ];
59
60      D22 = [ zeros(pp,mp)      ];
61
62      A_mat = A;
63      B_mat = [B1    B2];
64      C_mat = [C1;
65               C2];
66      D_mat = [D11    D12;
67               D21    D22];
68
69      AugG = ss(A_mat, B_mat, C_mat, D_mat);
70
71  end
```

Appendix E

```
1
2
3  #ifndef ISM_MSC_DEFINITIONS_H
4  #define ISM_MSC_DEFINITIONS_H
5
6  #define MAX_ROWS    4
7  #define MAX_COLS    4
8
9  //#define M_PI     3.141592653589793
10 #define RAD2RPM  9.5492965855       // 60/(2*M_PI)
11 #define RPM2RAD  0.1047197551       // (2*M_PI)/60
12
13 #define PWMMax          1911.0
14 #define PWMMin          1119.0
15
16 #define RAD2DEG         57.2957795131   // 180.0/M_PI
17 #define DEG2RAD         0.01745329252   // M_PI/180.0
18
19 #define INV_ROLL_PLANT  0.015654156   // 1 / Plant
20 #define INV_PITCH_PLANT 0.015654156   // 1 / Plant
21
22 void pwm2mxFunc(float *u, float *y);
23 void u2pwmFunc(float *u, float *y);
24
25 struct Matrix_Type{
26     unsigned int Rows, Cols;
27     float element[MAX_ROWS][MAX_COLS];
28 };
29
30 /*********************************************************
31  *            Classical Control                          *
32  *********************************************************/
33 static int16_t ISM_get_stabilize_roll(int32_t target_angle);
34 static int16_t ISM_get_stabilize_pitch(int32_t target_angle)
      ;
35
36 static int16_t ISM_get_rate_roll(float target_rate);
37 static int16_t ISM_get_rate_pitch(float target_rate);
38
39 /*********************************************************
40  *               Robust Control                          *
41  *********************************************************/
42
43 void ISM_Hinf_rate(float tRR, float tPR);
44 void ISM_Hinf_angle(int32_t target_roll, int32_t
```

```
    target_pitch);
45
46  void MatAdd(Matrix_Type *A, Matrix_Type *B, Matrix_Type *C);
                                // mat.C =   mat.A + mat.B
47  void MatMult(Matrix_Type *A, Matrix_Type *B, Matrix_Type *C)
    ;                           // mat.C =   mat.A x mat.B
48  void MatMult(float Aa, Matrix_Type *B, Matrix_Type *C);
                                    // mat.C = const.A x mat.B
49  void MatSet(Matrix_Type *A, unsigned int rows, unsigned int
    cols, float init);      // set matrix dimensions with a
    value for all elements
50
51  /* Updated values for:
52   *              b = 9.2737e-006
53   *              d = 2.57e-7
54   */
55  float tau2omega[4][4]={{173922.2991693123   ,  0.0 ,
    -972762.6459143970 , 26957.9563712434 },
56  {0.0 , -173922.2991693123   , 972762.6459143968 ,
    26957.9563712434 },
57  {-173922.2991693123 , 0.0 , -972762.6459143968 ,
    26957.9563712434 },
58  {0.0 ,  173922.2991693123   , 972762.6459143970 ,
    26957.9563712434 }};
59
60  float mx2cmdMat[4][4] ={{ 50.0 , 0.0 , -50.0 , 0.0},
61  {  0.0 , -50.0 ,   0.0 , 50.0},
62  {-25.0 ,  25.0 , -25.0 , 25.0},
63  { 25.0 ,  25.0 ,  25.0 , 25.0}};
64
65
66  /*****************************************************
67   *     Hinf Controllers (State-Space Form)           *
68   *****************************************************/
69
70  // States of Roll-Rate controller
71  Matrix_Type X_RR = { 4,          // Rows
72      1,           // Cols
73      { {0, 0, 0, 0},
74      {0, 0, 0, 0},
75      {0, 0, 0, 0},
76      {0, 0, 0, 0}
77      }
78  };
79
80  // States of Pitch-Rate controller
81  Matrix_Type X_PR = { 4,          // Rows
82      1,           // Cols
83      { {0, 0, 0, 0},
84      {0, 0, 0, 0},
85      {0, 0, 0, 0},
86      {0, 0, 0, 0}
87      }
88  };
89
90  Matrix_Type A_Hinf = { 4,  // Rows
91      4,  // Cols
92      {
```

```
93      { -50.038265399742   , -0.484494402658   ,
           -0.135473949527   ,       -1.815604517409   },
94      {                  0   , -0.006449149890   ,
           0   ,         0.000323567381   },
95      {4976.509980343927   , -0.426816948461   ,
           -10.135473949527   ,    -1461.493867488556   },
96      {                  0   ,  0.003704413240   ,
           1.000000000000   ,      -54.031215730544   }
97      }
98  };
99
100 Matrix_Type B_Hinf = { 4,                  // Rows
101     1,           // Cols
102     {
103     {                     0 , 0.0 , 0.0 , 0.0 },
104     {      -706.4436256228 , 0.0 , 0.0 , 0.0 },
105     {   -102812.9796354660 , 0.0 , 0.0 , 0.0 },
106     {     -3805.7366181715 , 0.0 , 0.0 , 0.0 }
107     }
108 };
109
110 Matrix_Type C_Hinf = { 1,                  // Rows
111     4,           // Cols
112     {
113     { 7.065426884076304 , -0.000687863541166   ,
           -0.000192339870484       , -0.002577714305571     },
114     { 0.0 , 0.0 , 0.0 , 0.0 },
115     { 0.0 , 0.0 , 0.0 , 0.0 },
116     { 0.0 , 0.0 , 0.0 , 0.0 }
117     }
118 };
119
120 Matrix_Type D_Hinf = { 1,                  // Rows
121 1,               // Cols
122 {
123 { 0.0 , 0.0 , 0.0 , 0.0 },
124 { 0.0 , 0.0 , 0.0 , 0.0 },
125 { 0.0 , 0.0 , 0.0 , 0.0 },
126 { 0.0 , 0.0 , 0.0 , 0.0 }
127 }
128 };
129
130 #endif
```

```
1
2   #include "ISM_Msc_Definitions.h"
3
4   /*****************************
5    * Note:
6    *              The priliminary controllers are using "
        pi_loiter_rate_lat/lon" and "pi_loiter_lat/lon"
        controller structures.
7    *              This mode should not be used for this
        software version.
8    */
9
10  void pwm2mxFunc(float *u, float *y){
11
12          float pwmP[4]={0.0 , 0.0 , 0.0 , 0.0};       // pwm
```

```
            of units % (Percentage)
13          uint16_t i,t;
14
15          float cmdOut[4]={0.0 , 0.0 , 0.0 , 0.0};
16
17          for(i=0 ; i<4 ; i++){
18              pwmP[i] = (u[i]-((PWMMin+PWMMax)/2.0))/(PWMMax-
                    PWMMin);
19          }
20
21          for(i=0 ; i<4 ; i++){
22              for(t=0 ; t<4 ; t++){
23                  cmdOut[i] += mx2cmdMat[i][t]*pwmP[t];
24              }
25          }
26          cmdOut[3]+=50.0;
27
28          for(i=0 ; i<4 ; i++){
29              y[i] = cmdOut[i];
30          }
31  }
32
33  void u2pwmFunc(float *u, float *y){
34
35          float pwmP[4];              // pwm of units % (
                Percentage)
36          uint16_t i,t;
37
38          float pwmOut[4]={0.0 , 0.0 , 0.0 , 0.0};
39          float omega2[4]={0.0 , 0.0 , 0.0 , 0.0};
40
41          for(i=0 ; i<4 ; i++){
42                  for(t=0 ; t<4 ; t++){
43                          omega2[i] += tau2omega[i][t]*u[t];
44                  }
45          }
46
47          for(i=0 ; i<4 ; i++){
48                  if(omega2[i]<0){
49                          omega2[i]=0;
50                  }
51                  omega2[i]=sqrt(omega2[i])*RAD2RPM;
52                  pwmOut[i]=(omega2[i
                        ]/6720.0+0.882455357142857)*1000.0;
53          }
54
55          for(i=0 ; i<4 ; i++){
56                  y[i] = pwmOut[i];
57          }
58
59  }
60
61  static int16_t ISM_get_stabilize_roll(int32_t target_angle){
62
63          // angle error
64          int16_t dummy = 0;
65
66          static APM_PI RollController;
```

```
67
68          RollController.kP(g.pi_loiter_lat.kP());
69          RollController.kI(0.0);
70
71          int32_t tmp_eAngle      = wrap_180(target_angle -
               ahrs.roll_sensor);     // eAngle = desire -
               measured : in deg*100
72          float eAngle            = (float)tmp_eAngle/100.0*
               DEG2RAD;
73
74          float targetRate= RollController.get_p(eAngle, dummy
               );
75
76          return ISM_get_rate_roll(targetRate);
77  }
78
79  static int16_t ISM_get_rate_roll(float targetRate){
80
81          static int32_t last_rate = 0;    // previous
               iterations rate
82          float p,i,d;                    // used to capture pid
                values for logging
83          int32_t current_rate;      // this iteration's rate
84          int32_t rate_d;                     // roll's
               acceleration
85          float eRate;                        // simply
               target_rate - current_rate
86          float output;                       // output from pid
                controller
87          float u[4];                                 // ISM:
               used as inputs for ISM functions
88          float y[4];                                 // ISM:
               used as outputs for ISM functions
89
90          int16_t dummy = 0;
91
92          Vector3f gyro = imu.get_gyro();
93
94          eRate = targetRate - (float)gyro.x;
95
96          p = g.pid_loiter_rate_lat.get_p(eRate, dummy);
97          i = g.pid_loiter_rate_lat.get_i(eRate, G_Dt, dummy);
98          d = g.pid_loiter_rate_lat.get_d(eRate, G_Dt, dummy);
99
100         output = p + i + d;
101
102         output = (float)output * INV_ROLL_PLANT;  //
               commanded tau (N.m)
103
104         // constrain output
105         output = constrain(output, -5.0, 5.0);
106
107         u[0] = output;     // u[0] : tau_phi command (N.m)
108         u[1] = 0;
109         u[2] = 0;
110         u[3] = 11.0;
111
112         u2pwmFunc(u, y);
```

```
113
114         u[0]=y[0];
115         u[1]=y[1];
116         u[2]=y[2];
117         u[3]=y[3];
118         pwm2mxFunc(u, y);
119
120         // output control, y[0] is ArduCopter phi command
121         return (int16_t)((float)y[0]*100.0);
122 }
123
124 static int16_t ISM_get_stabilize_pitch(int32_t target_angle)
        {
125
126         // angle error
127         int16_t dummy = 0;
128
129         static APM_PI PitchController;
130
131         PitchController.kP(g.pi_loiter_lat.kP());
132         PitchController.kI(0.0);
133
134         int32_t tmp_eAngle      = wrap_180(target_angle -
              ahrs.pitch_sensor);   // eAngle = desire -
              measured : in deg*100
135         float eAngle            = (float)tmp_eAngle/100.0*
              DEG2RAD;
136
137         float targetRate= PitchController.get_p(eAngle,
              dummy);
138
139         return ISM_get_rate_pitch(targetRate);
140 }
141
142 static int16_t ISM_get_rate_pitch(float targetRate){
143
144         static int32_t last_rate = 0;  // previous
              iterations rate
145         float p,i,d;                   // used to capture pid
              values for logging
146         int32_t current_rate; // this iteration's rate
147         int32_t rate_d;                // roll's acceleration
148         float eRate;                   // simply target_rate -
              current_rate
149         float output;                  // output from pid
              controller
150         float u[4];                            // ISM: used as
              inputs for ISM functions
151         float y[4];                            // ISM: used as
              outputs for ISM functions
152
153         Vector3f gyro = imu.get_gyro();
154
155         int16_t dummy = 0;
156
157         eRate           = targetRate - (float)gyro.y;
158
159         p = g.pid_loiter_rate_lon.get_p(eRate, dummy);
```

```
160         i = g.pid_loiter_rate_lon.get_i(eRate, G_Dt, dummy);
161         d = g.pid_loiter_rate_lon.get_d(eRate, G_Dt, dummy);
162         output = p + i + d;
163
164         output          = (float)output * INV_PITCH_PLANT;
                              // commanded tau (N.m)
165
166         // constrain output
167         output = constrain(output, -5.0, 5.0);
168
169         u[0] = 0;                                   // u[0] :
               tau_phi command (N.m)
170         u[1] = output;
171         u[2] = 0;
172         u[3] = 11.0;
173
174         u2pwmFunc(u, y);
175
176         u[0]=y[0];
177         u[1]=y[1];
178         u[2]=y[2];
179         u[3]=y[3];
180         pwm2mxFunc(u, y);
181
182         // output control, y[1] is ArduCopter pitch command
183         return (int16_t)((float)y[1]*100.0);
184 }
185
186 void ISM_Hinf_angle(int32_t target_roll, int32_t
       target_pitch){
187
188         // angle error
189         int16_t dummy = 0;
190
191         static APM_PI RollController;
192         static APM_PI PitchController;
193
194         RollController.kP(g.pi_loiter_lat.kP());
195         RollController.kI(0.0);
196
197         PitchController.kP(g.pi_loiter_lon.kP());
198         PitchController.kI(0.0);
199
200         int32_t tmp_eRoll       = wrap_180(target_roll -
               ahrs.roll_sensor);              // eAngle = desire
                - measured : in deg*100
201         int32_t tmp_ePitch      = wrap_180(target_pitch -
               ahrs.pitch_sensor);   // eAngle = desire -
               measured : in deg*100
202
203         float eRoll    = (float)tmp_eRoll/100.0*DEG2RAD;
204         float ePitch   = (float)tmp_ePitch/100.0*DEG2RAD;
205
206         float targetRollRate    = RollController.get_p(eRoll
               , dummy);
207         float targetPitchRate   = PitchController.get_p(
               ePitch, dummy);
208
```

```
209         ISM_Hinf_rate(targetRollRate, targetPitchRate);
210
211 }
212
213 void ISM_Hinf_rate(float tRR, float tPR){
214
215         int32_t current_RR;  // this iteration's roll rate
216         int32_t current_PR;  // this iteration's pitch rate
217         int32_t RR_d;        // roll's acceleration
218         int32_t PR_d;        // pitch's acceleration
219         float eRR;           // simply target_rate -
                current_rate
220         float ePR;           // simply target_rate -
                current_rate
221         float tauPhi;        // Tau_Phi (roll torque command
                )
222         float tauTheta;      // Tau_Theta (pitch torque
                command)
223         float u[4];          // ISM: used as inputs for ISM
                functions
224         float y[4];          // ISM: used as outputs for ISM
                 functions
225
226         Vector3f gyro = imu.get_gyro();
227
228     Matrix_Type tmp1, tmp2, tmp3;
229     Matrix_Type *A, *B, *C, *D, *XRR, *XPR;
230     Matrix_Type dX;
231
232         eRR                     = tRR - (float)gyro.x;
233         ePR                     = tPR - (float)gyro.y;
234
235     A = &A_Hinf;
236     B = &B_Hinf;
237     C = &C_Hinf;
238     D = &D_Hinf;
239
240     XRR = &X_RR;
241     XPR = &X_PR;
242
243     //------------ Roll Rate Ctrl Propagation ---------
244
245     MatMult(A     ,  XRR   , &tmp1);     // tmp1 = A . X
246     MatMult(eRR   ,  B     , &tmp2);     // tmp2 = B . U
247     MatAdd(&tmp1  , &tmp2  , &dX);       // dX   = tmp1 + tmp2
248
249     MatMult(G_Dt  , &dX    , &tmp1);     // tmp1 = dX . Ts
250     MatAdd(&tmp1  ,  XRR   , &tmp2);     // tmp2 = X + tmp1
251     *(XRR) = tmp2;
252
253     MatMult( C    ,  XRR   , &tmp1);     // tmp1 = C . X
254     MatMult(eRR   ,  D     , &tmp2);     // tmp2 = D . U
255     MatAdd(&tmp1  , &tmp2  , &tmp3);     // tmp3 = tmp1 + tmp2
256
257     u[0] = tmp3.element[0][0] * INV_ROLL_PLANT;
                // tau_phi command
258
259     //----------------------------- Pitch Rate Ctrl
```

```
        Propagation -------------
260
261     MatMult(A      ,  XPR   , &tmp1);     // tmp1 = A . X
262     MatMult(ePR ,  B     , &tmp2);     // tmp2 = B . U
263     MatAdd(&tmp1  , &tmp2 , &dX);       // dX   = tmp1 + tmp2
264
265     MatMult(G_Dt  , &dX   , &tmp1);     // tmp1 = dX . Ts
266     MatAdd(&tmp1  ,  XPR  , &tmp2);     // tmp2 = X + tmp1
267     *(XPR) = tmp2;
268
269     MatMult( C    ,  XPR  , &tmp1);     // tmp1 = C . X
270     MatMult(ePR   ,  D    , &tmp2);     // tmp2 = D . U
271     MatAdd(&tmp1  , &tmp2 , &tmp3);     // tmp3 = tmp1 + tmp2
272
273     u[1] = tmp3.element[0][0] * INV_PITCH_PLANT;
                          // tau_theta command
274
275         u[2] = 0;
276         u[3] = 11.0;
277
278         u2pwmFunc(u, y);
279
280         u[0]=y[0];
281         u[1]=y[1];
282         u[2]=y[2];
283         u[3]=y[3];
284
285         pwm2mxFunc(u, y);
286
287         g.rc_1.servo_out = (int16_t)((float)y[0]*100.0);
                             // y[0] is the roll command
288         g.rc_2.servo_out = (int16_t)((float)y[1]*100.0);
                             // y[1] is the pitch command
289
290 }
291
292 void MatMult(Matrix_Type *A, Matrix_Type *B, Matrix_Type *C)
        {
293     C->Rows=A->Rows;
294     C->Cols=B->Cols;
295
296     float tmp;
297
298     for(unsigned int i=0;i<C->Rows;i++){
299         for(unsigned int t=0;t<C->Cols;t++){
300             tmp=0;
301             for(unsigned int k=0;k<A->Cols;k++){
302                 tmp = tmp + A->element[i][k]*B->element[k][t
                        ];
303             }
304             C->element[i][t]=tmp;
305         }
306     }
307 }
308
309 void MatAdd(Matrix_Type *A, Matrix_Type *B, Matrix_Type *C){
310     C->Rows=A->Rows;
311     C->Cols=B->Cols;
```

```
312
313     for(unsigned int i=0;i<C->Rows;i++){
314         for(unsigned int t=0;t<C->Cols;t++){
315             C->element[i][t]=A->element[i][t]+B->element[i][
                    t];
316         }
317     }
318 }
319
320 void MatMult(float K, Matrix_Type *A, Matrix_Type *C){
321     C->Rows=A->Rows;
322     C->Cols=A->Cols;
323     for(unsigned int i=0;i<A->Rows;i++){
324         for(unsigned int t=0;t<A->Cols;t++){
325             C->element[i][t]=K*A->element[i][t];
326         }
327     }
328 }
329
330 void MatSet(Matrix_Type *A, unsigned int rows, unsigned int
        cols, float init){
331     A->Rows=rows;
332     A->Cols=cols;
333     for(unsigned int i=0;i<A->Rows;i++){
334         for(unsigned int t=0;t<A->Cols;t++){
335             A->element[i][t]=init;
336         }
337     }
338 }
```

Appendix F

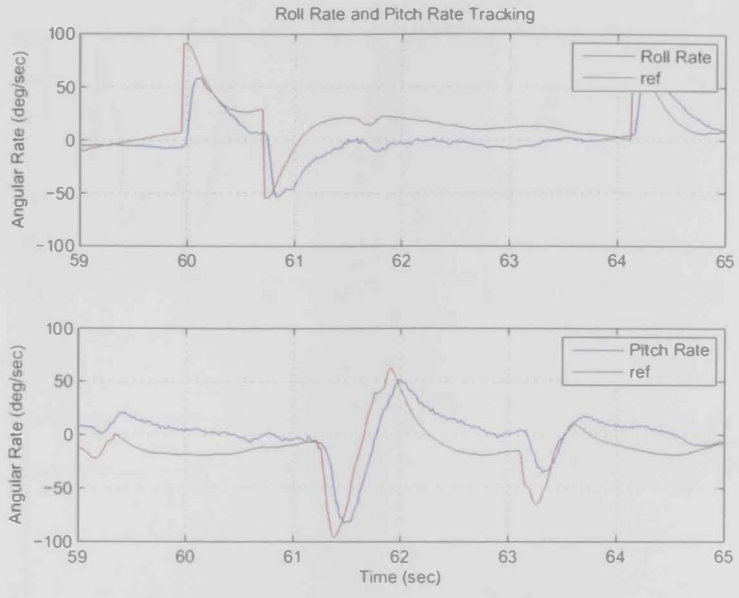
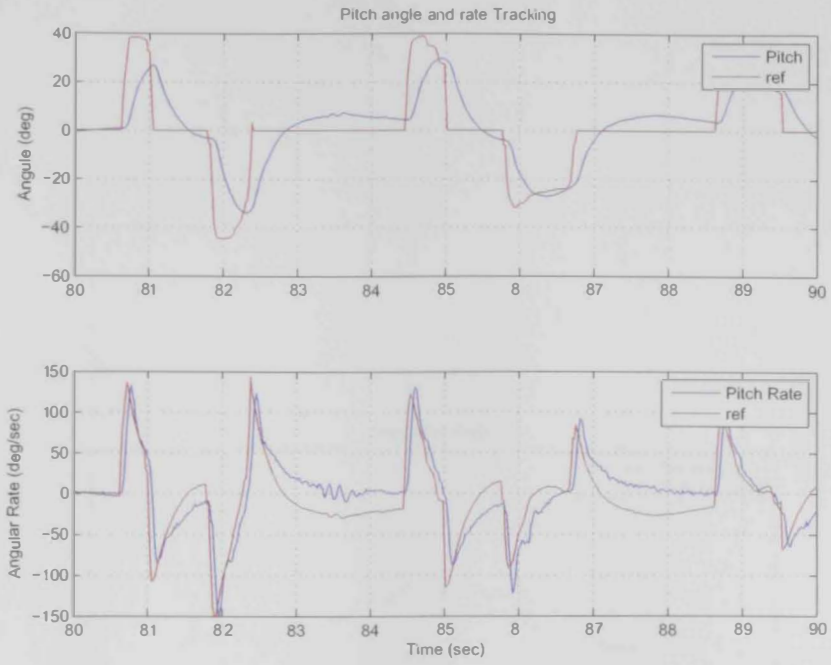
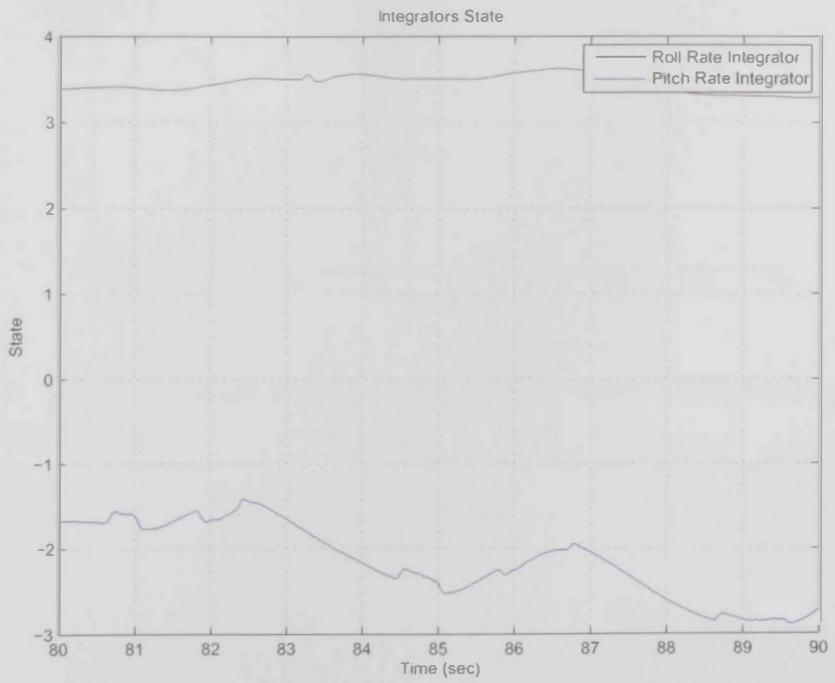


Figure 1: Steady-state error in PID controller with no integration action



(a) Tracking



(b) Integrators

Figure 2: Return of steady-state error on pitch after exciting the system and the integrator state changing to an untrimmed value

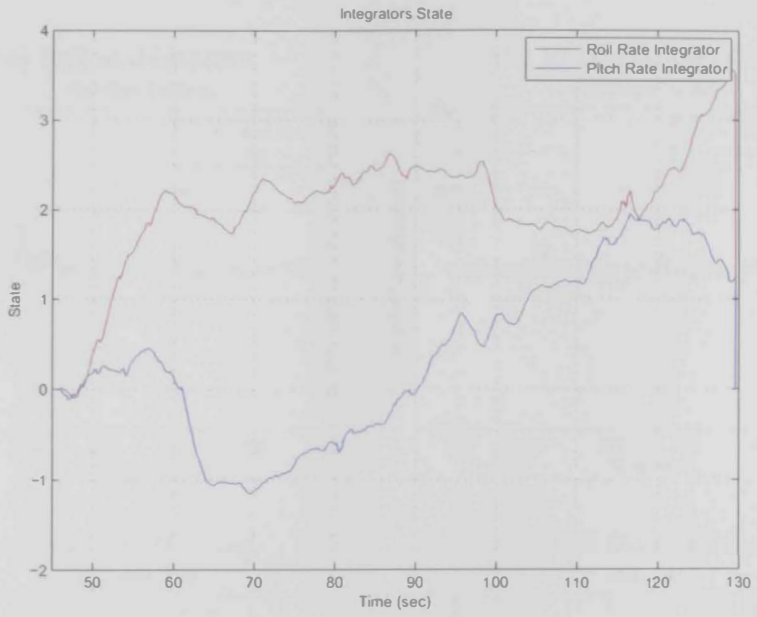
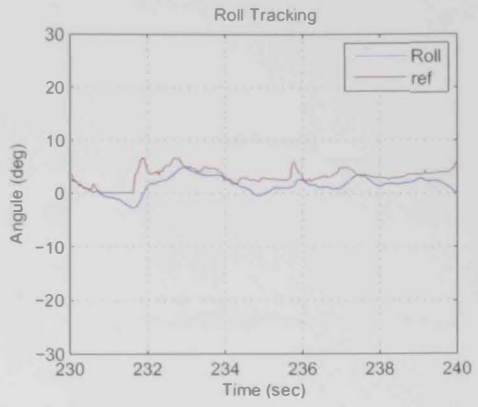
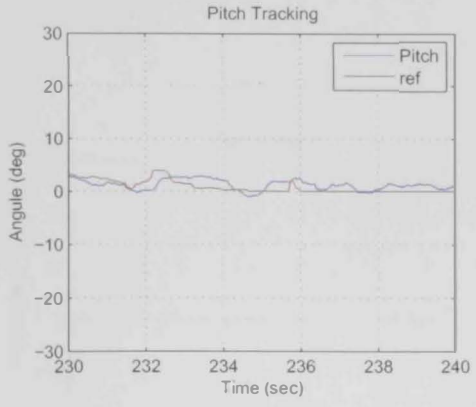


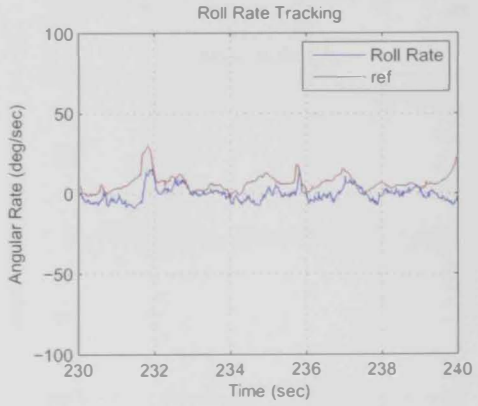
Figure 3: Integrator state change in non-steady wind (changing magnitude and direction)



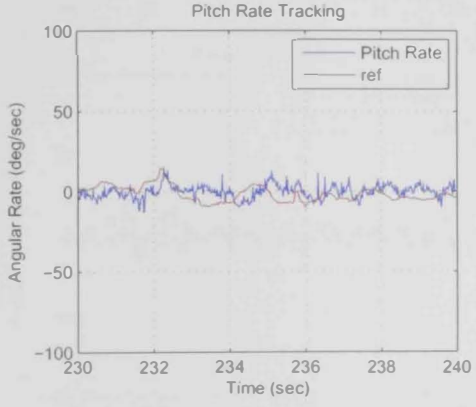
(a) Roll stabilization.



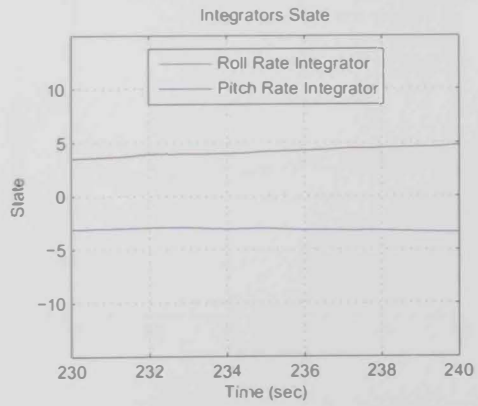
(b) Pitch stabilization.



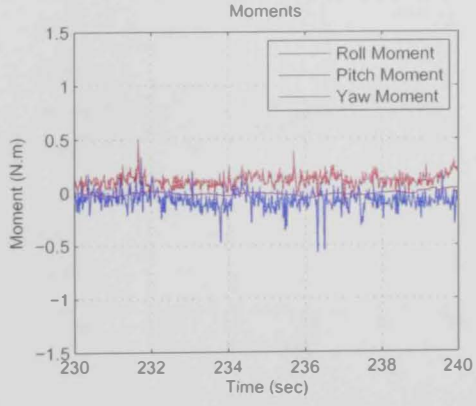
(c) Roll rate tracking.



(d) Pitch Rate tracking.



(e) Integrators' state.



(f) Moments.

Figure 4: System stabilization and tracking in steady wind for PID controller

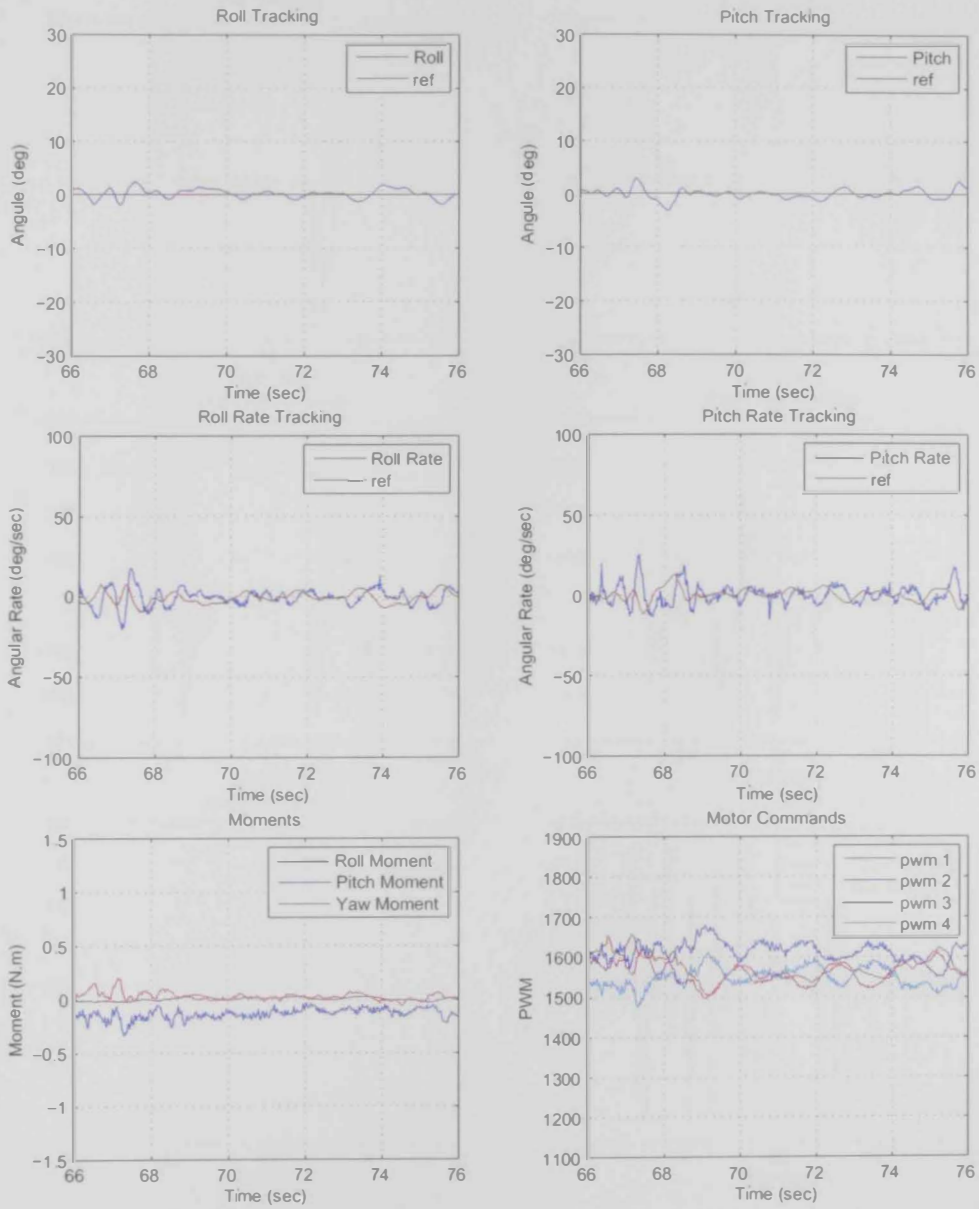


Figure 5: System stabilization and tracking in steady wind for robust controller

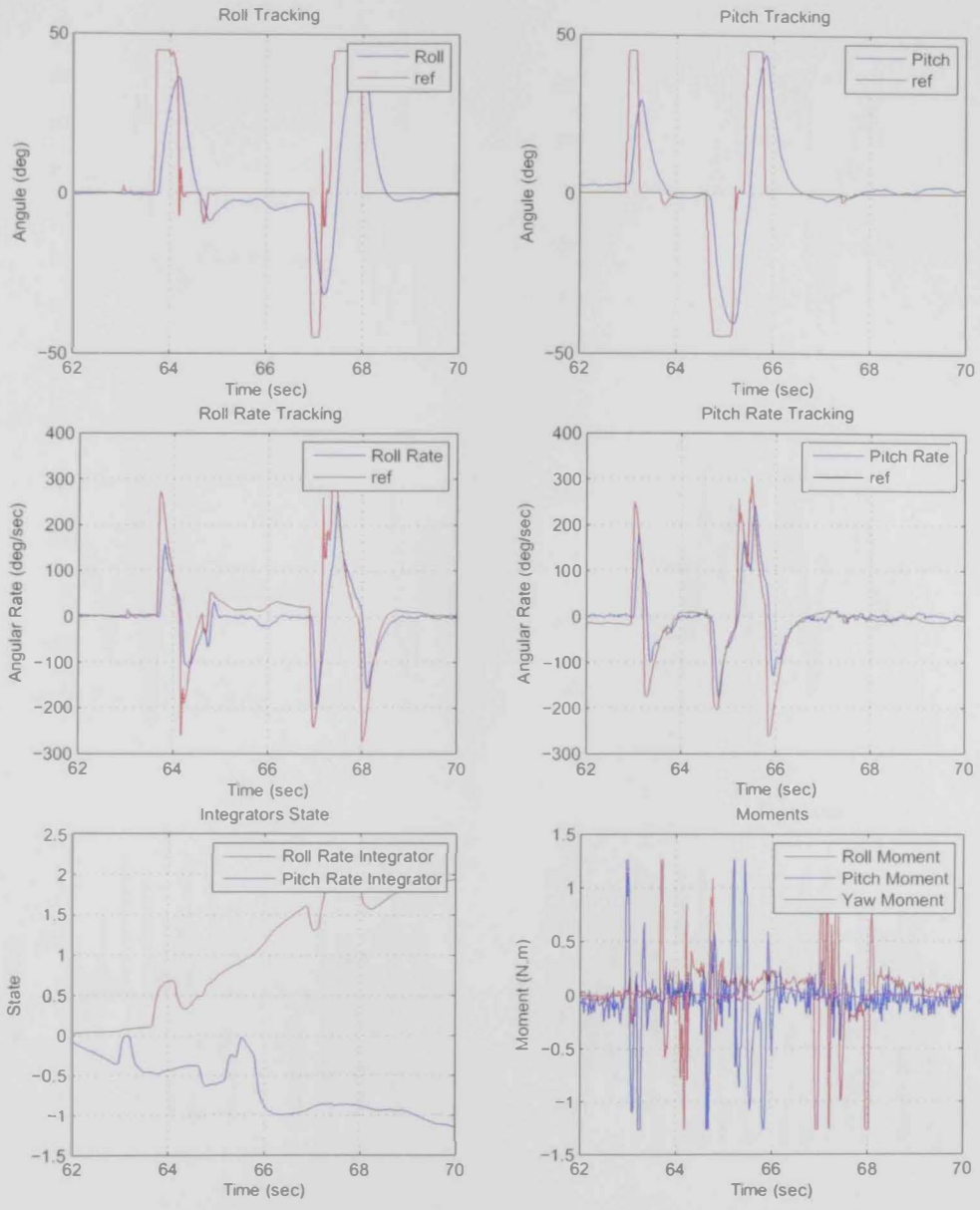


Figure 6: System stabilization and tracking for P attitude controller with high gain ($K_p = 6$), PID controller for angular rate loop

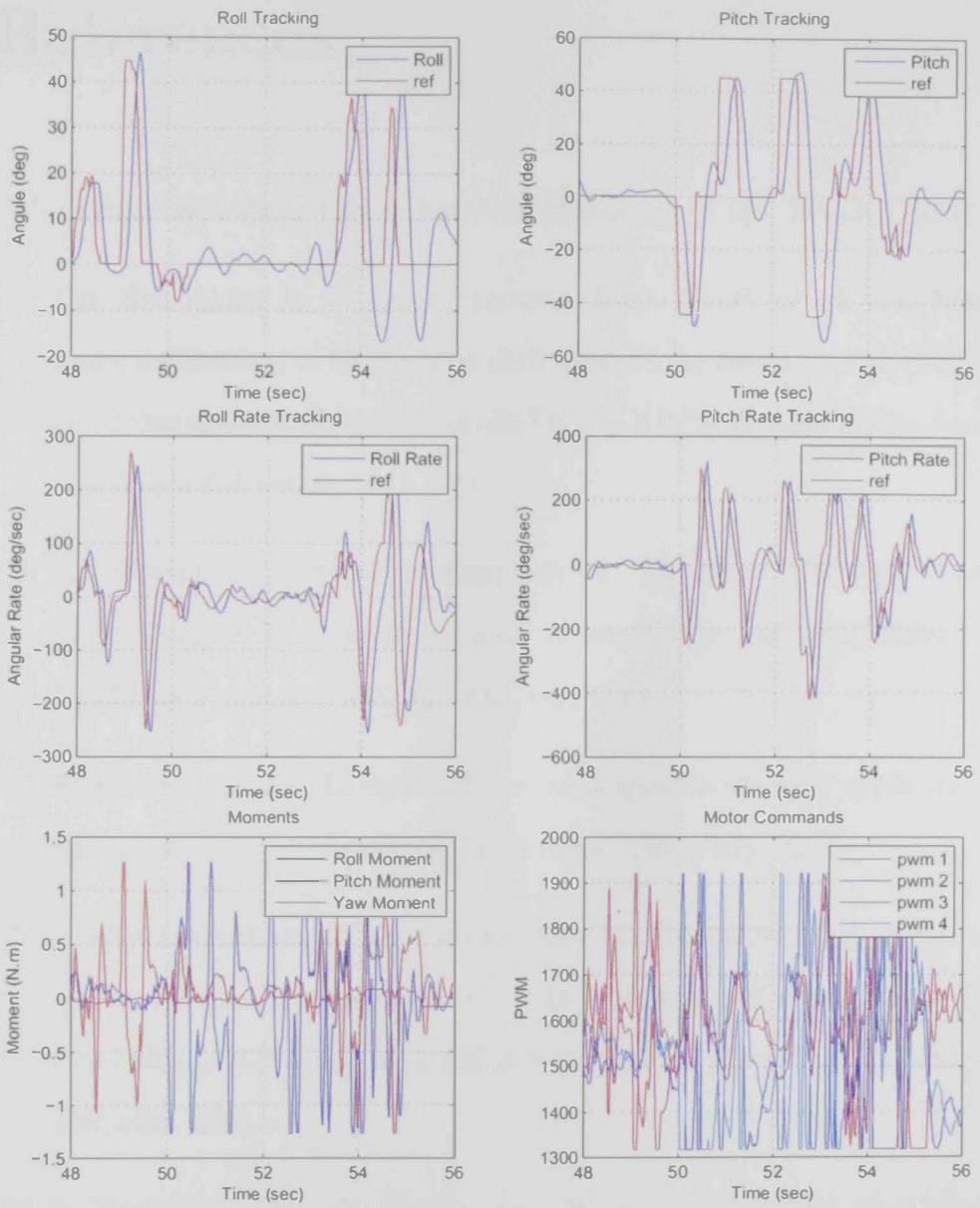


Figure 7: System stabilization and tracking for P attitude controller with high gain ($K_p = 6$), robust controller for angular rate loop

References

[1] Flight No. 740. The de bothezat helicopter. *p 125*, March 1, 1923.

[2] Ch. Sai Babu A. Purna Chandra Rao, Y. P. Obulesh. Mathematical modeling of bldc motor with closed loop speed control using pid controller under various loading conditions. *ARPN Journal of Engineering and Applied Sciences*, 2012.

[3] A. Benallegue, A. Mokhtari, and L. Fridman. High order sliding mode observer for a quadrotor uav. *International Journal of Robust and Nonlinear Control*, **18**[45]:427–440, mar 2008.

[4] S. Bouabdallah. Design and control of quadrotors with application to autonomous flying. *Lausanne Polytechnic University*, 2007.

[5] S. Bouabdallah, P. Murrieri, and R. Siegwart. Design and control of an indoor micro quadrotor. In *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, **5**, page 43934398, 2004.

[6] S. Bouabdallah, A. Noth, and R. Siegwart. Pid vs lq control techniques applied to an indoor micro quadrotor. **3**:2451–2456 vol.3, oct 2004.

[7] S. Bouabdallah and R. Siegwart. Full control of a quadrotor. page 153158, nov 2007.

[8] M. Tayebi A. Mullhaupt P. Bouchoucha, M. Tadjine. Step by step robust nonlinear pi for attitude stabilisation of a four-rotor mini-aircraft. *16th Mediterranean Conference on Control and Automation*, 2008.

[9] Pedro Castillo, Rogelio Lozano, and Alejandro E. Dzul. *Modelling and control of mini-flying machines*. Springer, 2005.

[10] Urs Christen. *Engineering Aspects of H_∞ Control*. PhD thesis, Swiss Federal Institute Of Technology, Diss. ETH No. 11433, 1996.

[11] cole centrale Paris. (french) les grands centraux : tienne hmichen. *Centrale-Histoire*, 1884-1955.

[12] C. Coza and C. J.B Macnab. A new robust adaptive-fuzzy control method applied to quadrotor helicopter stabilization. page 454458, jun 2006.

[13] Marcelo De Lellis Costa de Oliveira. Modeling, identification and control of a quadrotor aircraft. 2012.

[14] M. O Efe. Robust low altitude behavior control of a quadrotor rotorcraft through sliding modes. page 16, jun 2007.

[15] Z. Fang, X. Wang, and J. Sun. Design and nonlinear control of an indoor quadrotor flying robot. In *Intelligent Control and Automation (WCICA), 2010 8th World Congress on*, page 429434, 2010.

[16] Flight. Convertawings inc. *P 722-723*, November 1956.

[17] Rahul Goel, Sapan Shah, Nitin Gupta, and N. Ananthkrishnan. Modeling, simulation and flight testing of an autonomous quadrotor. In *ICEAE*, 2009.

[18] Petkov Petko H. Konstantinov Mihail M Gu, Da-Wei. *Robust Control Design with MATLAB*. Springer, 2005.

[19] G. Hoffmann, D. G Rajnarayan, S. L Waslander, D. Dostal, J. S Jang, and C. J Tomlin. The stanford testbed of autonomous rotorcraft for multi agent control (starmac). In *Digital Avionics Systems Conference, 2004. DASC 04. The 23rd*, 2, page 12E, 2004.

[20] G. M Hoffmann, H. Huang, S. L Waslander, and C. J Tomlin. Quadrotor helicopter flight dynamics and control: Theory and experiment. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, page 120, 2007.

[21] G. M Hoffmann, S. L Waslander, and C. J Tomlin. Quadrotor helicopter trajectory tracking control. In *Proceedings of the AIAA Guidance, Navigation, and Control Conference*, 2008.

[22] Brandon W. Gordon Hojjat A. Izadi, Youmin Zhang. Fault tolerant model predictive control of quad-rotor helicopters with actuator fault estimation. *IFAC World Congress 18th*, 2011.

[23] Youmin Zhang Camille-Alain Rabbath Iman Sadeghzadeh, Ankit Mehta. Fault-tolerant trajectory tracking control of a quadrotor helicopter using gain-scheduled pid and model reference adaptive control. *Annual Conference of the Prognostics and Health Management Society*, 2011.

[24] M. P. Miller. An accurate method of measuring the moments of inertia of airplanes. Technical report, National Advisory Committee for Aeronatics, 1930.

[25] V. Mistler, A. Benallegue, and NK M'Sirdi. Exact linearization and noninteracting control of a 4 rotors helicopter via dynamic feedback. In *Robot and Human Interactive Communication, 2001. Proceedings. 10th IEEE International Workshop on*, page 586593, 2001.

[26] A. Mokhtari and A. Benallegue. Dynamic feedback controller of euler angles and wind parameters estimation for a quadrotor unmanned aerial vehicle. 3:2359 2366 Vol.3, may 2004.

[27] A. Mokhtari, A. Benallegue, and B. Daachi. Robust feedback linearization and GH controller for a quadrotor unmanned aerial vehicle. page 1198 1203, aug 2005.

[28] A. Mokhtari, A. Benallegue, and Y. Orlov. Exact linearization and sliding mode observer for a quadrotor unmanned aerial vehicle. *Int. J. Robot. Autom.*, **21**[1]:3949, jan 2006.

[29] C. Nicol, C. J.B Macnab, and A. Ramirez-Serrano. Robust neural network control of a quadrotor helicopter. page 001233001238, may 2008.

[30] S. Park, D. H Won, M. S Kang, T. J Kim, H. G Lee, and S. J Kwon. Ric (robust internal-loop compensator) based flight control of a quad-rotor type uav. page 3542 3547, aug 2005.

[31] U. Pilz, A. P Popov, and H. Werner. Robust controller design for formation flight of quad-rotor helicopters. page 83228327, dec 2009.

[32] Timothy W. McLain Randal W. Beard. *Small Unmanned Aircraft - Theory and Practice*. Princeton University, 2012.

[33] I. I. I. Shepherd and Kagan Tumer. Robust neuro-control for a micro quadrotor. In *Proceedings of the 12th annual conference on Genetic and evolutionary computation*, GECCO '10, page 11311138, New York, NY, USA, 2010. ACM. ACM ID: 1830693.

[34] Young S. Suh. Robust control of a quad-rotor aerial vehicle. *International Journal of Applied Electromagnetics and Mechanics*, 2003.

[35] OLIVER TANNER. *Modeling, Identification, and Control of Autonomous Helicopters*. PhD thesis, Swiss Federal Institute Of Technology, Diss. ETH No. 14899, 2003.

[36] A. TASHAKORI. Modeling of bldc motor with ideal back-emf for automotive applications. *Proceedings of the World Congress on Engineering*, 2011.

[37] A. TAYEBI AND S. MCGILVRAY. Attitude stabilization of a vtol quadrotor aircraft. *Control Systems Technology, IEEE Transactions on*, 14[3]:562571, 2006.

[38] A. TAYEBI AND M. B ZAREMBA. Internal model-based robust iterative learning control for uncertain LTI systems. In *Decision and Control, 2000. Proceedings of the 39th IEEE Conference on*, 4, page 34393444, 2000.

[39] A. TAYEBI AND M. B. ZAREMBA. Robust iterative learning control design is straightforward for uncertain LTI systems satisfying the robust performance condition. *Automatic Control, IEEE Transactions on*, 48[1]:101106, 2003.

[40] "HTTP://WWW.FLICKR.COM/PHOTOS/AMPHALON/6099854659/IN/PHOTOSTREA TOME WIGLEY.

[41] Q. L ZHOU, Y. ZHANG, C. A RABBATH, AND D. THEILLIOL. Design of feedback linearization control and reconfigurable control allocation with application to a quadrotor UAV. In *Control and Fault-Tolerant Systems (SysTol), 2010 Conference on*, page 371376, 2010.

، ولكنه يفتقر إلى الصلابة لعدم وجود التطابق بين المحركات ومراوح النظام. ولقد تم إضافة الفعل المتكامل بمقدار كاف للمتحكم التقليدي ، بحيث يتحسن أداء المتحكم التقليدي ولا تتأثر ردة الفعل الديناميكية للمتحكم بشكل كبير. في الصورة النهائية للمتحكم ، أعطى المتحكم التقليدي أداء وقوة مقبولين وايضا تحمل مقبول للإضطرابات المدرجة. و من جانب اخر ، كان أداء المتحكم القوي أفضل بكثير من المتحكم التقليدي من ناحية ردة الفعل الديناميكية وتحمل الإضطرابات ، ولكن المتحكم القوي معقد في أساسة بشكل أكبر بكثير من المتحكم التقليدي ، ولذلك يتطلب قوة حسابية أكبر.

في تمثيل النظام الواقعي. وبعد ذلك تليه خطوة تصميم نظام التحكم من خلال استخدام نوعين من المتحكمات ، وهما: المتحكم التقليدي PID و المتحكم القوي.

تم تصميم هيكل نظام التحكم لاستيعاب طائرات أخرى من نفس النوع و تم تصميم المتحكم التقليدي للسرعة الزاوية تحليليا. بينما تم تصميم المتحكم لوضعية الطائرة باستخدام الرسومات البيانية Root-Locus. اضافة على ذلك يقوم عمل المتحكم القوي على نظرية الH_∞ ، والي تعتمد على تخطيط GS/T في بناء النموذج الخطي المركب.

يتناول جانب من هذا العمل البحثي تغيير واعادة صياغة بعض البرمجيات المتوفرة للنظام Arducopter ، ومن اهم هذه التغييرات تخصيص البيانات المسجلة، تخصيص وظائف اوضاع الطيران الاصلية وتخصيص أجزاء من البرمجيات الأصلية . اضافة لما تم ذكره، يقوم العمل البحثي على إضافة تطبيق المتحكم القوي من خلال ما يسمى بـ(state-space).

وفي مراحل البحث الاخيرة ، تم إجراء عدة تحليقات تجريبية للطائرة لضبط و اختبارنظام المتحكم التقليدي PID واختبار المتحكم القوي ، بالاضافة إلى إجراء اختبارات القوة والأداء و التي تتم من خلال إدراج إضطرابات للنظام أثناء فترة التحليق ، بحيث تكون هذه الاضطرابات معلومة ويكون الطيار هو المتحكم بإدراجها من غير إدراك نظام التحكم.

لقد تم استنتاج العديد من النتائج خلال مراحل هذا البحث. وإحدى هذه النتائج هي فاعلية الطرق التي تم استخدمها في تقدير وإيجاد عوامل الطائرة بالرغم من القصور في التجارب المستقلة. اضافة إلى ذلك ، يكشف هذا البحث عن قصور نموذج العزم الدوراني حول محور الإتجاه المعاد استخدامه ، والذي تبين انه غير مطابق للنظام الواقعي.ويبين النموذج المعاد استخدامه ان الإحتكاك بين المراوح والهواء أثناء دورانهما هو العامل المهيمن على ردة الفعل العزمية ، و من جهة اخرى اظهرت البيانات المسجلة والتحاليل المقدمة في هذا البحث على ان ردة الفعل العزمية من قبل عزم المحرك هي المكون الأكبر والمهيمن على إحتكاك المراوح بالهواء في تشكيل لعزم الدوراني على محور الإتجاه.

و من جانب التحكم بالطيران التجريبي ، تم تطبيق المتحكمات PID و المتحكم القوي وتم اختبارهما في تحليقات مختلفة. وتظهر البيانات المسجلة والنتائج أن المتحكم التقليدي PD يستجيب بشكل ديناميكي جيد

مقدمة

العمل البحثي المقدم في هذه الرسالة يهدف الى تصميم وتحليل وتطوير نظام التحكم القوي لطائرة رباعية المراوح ، وتم استخدام طائرة الArducopter كمنصة للتطبيق والمبنية على هيكل من نوع ال3DR. المشروع البحثي يهدف ايضا الى تسليط الضوء على التعديلات التي تم اجرائها على نموذج النظام الرئيسي للطائرة و تحديد العوامل والخطوات اللازمة لتحقيق الهدف المرجو من هذا العمل البحثي.

العمل البحثي المقدم يستثني تحديد عامل التأثير الجايروسكوبي للمراوح،افتراضا على ان المعامل بسيط نسبيا، وايضا تم استثناء التطبيق المتحكم في اتجاه الطائرة والإرتفاع، بينما يناقش البحث موضوع التحكم في الاتجاه والإرتفاع من خلال التصميم وخاصية المحاكاه فقط. يعود استثناء التحكم بالإتجاه لسببين رئيسيين، السبب الأول هو قصور نموذج العزم في محور الإتجاه المعاد استخدامه عندما تمت المقارنة بين النموذج والطائرة والسبب الثاني هو معاناة نظام ArduCopter من خلل في التصميم، والذي يتسبب في انحراف مستمر للإتجاه المقدر. أما سبب استثناء التحكم في الإرتفاع من التطبيق فيعود إلى أن التحكم في الإرتفاع يتطلب تصميم وتطبيق لنظام ملاحة متكامل ، وهو غير متوفر حاليا.

ومن المساهمات المقدمة في هذا البحث، تحديد العوامل للنظام ، تصميم نظام تحكم قوي ، تغيير برمجيات النظام، التطبيق على المتحكم التقليدي PID و المتحكم القوي ، تزويد النظام الأصلي بقطع إضافية لإجراء إختبارات القوة والأداء وعرض وتحليل النتائج المستخلصة من هذه التجارب.

ويشتمل العمل في هذا البحث على قراءة و تحليل نتائج الأبحاث الاخرى في هذا المجال ، وتم إعادة استخدام هذه النتائج لتقديم اضافات جديدة في مجال التحكم بنظام الطائرة رباعية المراوح. لذلك ، يقوم العمل في هذا البحث على خطة عمل تبدأ ببناء نموذج كامل للنظام ، ومن ثم يتم تحصيل نموذج غير خطي للطائرة يكون صالحا لمرحلة السكون في مرحلة الطيران، ويتبعه تنفيذ هذا النموذج في برنامج الSimulink. إضافة إلى بناء النموذج ، يسعى العمل البحثي على تقدير وإيجاد عوامل الطائرة المستخدمة مثل: معامل القوة للمراوح ، القصور الذاتي للطائرة على جميع محاورها ، ومعامل الإحتكاك بين المراوح والهواء. حيث تعتمد عملية إيجاد العوامل في هذا البحث على تجارب مستقلة وبيانات مسجلة من طيران تجريبي ، ويتم التحقق من دقة النموذج الغير خطي والتأكد من صلاحية النموذج الافتراضي

جامعة الامارات العربية المتحدة

كلية الهندسة

قسم الهندسة الكهربائية

تحديد العوامل والتحكم القوي مطبقة على طائرة رباعية المراوح

إسماعيل سليمان آل علي

رسالة مقدمة لاستكمال متطلبات الحصول على درجة الماجستير في الهندسة الكهربائية

قسم الهندسة الكهربائية

باشراف د. حسن نورا

يونيو 2014